SmartShip Training Session

Ioannis Kontopoulos
Department of Informatics and Telematics, Harokopio University of Athens, Greece
e-mail: kontopoulos@hua.gr

January 2022

Contents

List of Figures

1 Introduction

During this training session, participants will get acquainted with a newly developed extension of the PostgreSQL database, called MobilityDB. MobilityDB is developed on top of the PostGIS extension which adds support for geographic objects allowing location queries to be run in SQL. MobilityDB is designed to be efficient for queries related to mobility data e.g., trajectories of moving objects. Furthermore, participants will use MobilityDB to query maritime tracking data originating from vessels and will get to know the AIS (Automatic Identification System), the system through which vessels transmit their positions periodically. While this workshop illustrates the usage of some MobilityDB functions, it does not explain them in detail. If you need help concerning the functions of MobilityDB, please refer to the documentation or contact the authors of the database [1, 2, 3, 4].

2 Requirements for the training session

The installations below are required for someone to run the queries that will be demonstrated in the training session. Furthermore, MobilityDB requires **Linux** to be run (other UNIX-like systems may work, but remain untested). Any distribution of **Ubuntu** >= **16.04** is a preferred operating system for this session. Participants can either run MobilityDB with docker (Section 2.1) **OR** they can install the database themselves (Section 2.2).

2.1 Run with docker

By running MobilityDB with docker, the only installation that is required is the docker engine. To install docker, simply type in the terminal window the following commands:

Listing 1: Install docker engine

```
sudo apt update
sudo apt install docker.io
```

The Docker service can be setup to run at startup. To do so, type in each command followed by enter:

Listing 2: Start and Automate Docker

```
sudo systemctl start docker
sudo systemctl start docker
```

To download a docker container that runs PostgreSQL-12, PostGIS-2.5 and MobilityDB-develop type the following commands:

Listing 3: Pull the prebuilt image of MobilityDB from the Docker Hub Registry

```
docker pull mobilitydb/mobilitydb:12-2.5-develop-workshop
```

Listing 4: Create a Docker volume to preserve the PostgreSQL database files outside of the container

```
docker volume create mobilitydb_data
```

Listing 5: Run the docker container (one command that takes up two lines)

```
docker run --name "mobilitydb" -d -p 5432 -v mobilitydb_data:/var/lib/postgresql \
mobilitydb/mobilitydb:12-2.5-develop-workshop
```

Listing 6: Enter into the Docker container

```
docker exec -it mobilitydb bash
```

Listing 7: Connect to the database (*username* = *docker*, *database* = *mobilitydb*)

```
psql -U docker -d mobilitydb
```

2.2 Run natively

To install MobilityDB, PostgreSQL and PostGIS== 2.5 need to be installed first. To install PostgreSQL 11, it is recommended to update the system packages:

Listing 8: Update system packages

```
sudo apt update && sudo apt -y upgrade
sudo reboot
```

Before adding repository content to Ubuntu 20.04/18.04/16.04, we need to import the repository signing key:

Listing 9: Add PostgreSQL 11 APT repository (1)

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc \
| sudo apt-key add -
```

After importing GPG key, add repository contents to Ubuntu 20.04/18.04/16.04:

Listing 10: Add PostgreSQL 11 APT repository (2)

```
RELEASE=$(lsb_release -cs)
echo "deb http://apt.postgresql.org/pub/repos/apt/ ${RELEASE}"-pgdg main | \
sudo tee /etc/apt/sources.list.d/pgdg.list
```

Verify repository file contents:

Listing 11: Add PostgreSQL 11 APT repository (3)

```
cat /etc/apt/sources.list.d/pgdg.list
deb http://apt.postgresql.org/pub/repos/apt/ bionic-pgdg main
```

The last installation step is for PostgreSQL 11 packages. Run the following commands to install PostgreSQL 11 on Ubuntu 20.04/18.04/16.04:

Listing 12: Install PostgreSQL 11

```
sudo apt update
sudo apt -y install postgresql-11
```

Listing 13: Set PostgreSQL admin user's password and do testing

```
sudo su - postgres
psql -c "alter user postgres with password 'StrongPassword'"
```

More details on the installation of PostgreSQL 11 can also be found here. To install PostGIS== 2.5 type in the terminal the following commad:

Listing 14: Install PostGIS== *2.5*

```
sudo apt-get install postgresql-11-postgis-2.5
```

Before installing MobilityDB to our system, further dependencies need to be installed. To install the dependencies type:

Listing 15: MobilityDB dependencies

```
sudo apt-get install libgsl-dev
sudo apt install build-essential cmake postgresql-server-dev-11 liblwgeom-dev \
libproj-dev libjson-c-dev
```

To built MobilityDB type:

Listing 16: Building and Installation

```
git clone https://github.com/MobilityDB/MobilityDB
mkdir MobilityDB/build
cd MobilityDB/build
cmake ..
make
sudo make install
psql -c 'CREATE EXTENSION MobilityDB CASCADE'
```

We should also set the following in postgresql.conf:

Listing 17: Configuration

```
shared_preload_libraries = 'postgis-2.5'
max_locks_per_transaction = 128
```

The configuration of PostgreSQL is typically located at '/etc/postgresql/9.1/main/postgresql.conf'. Installation instructions for MobilityDB can also be found here.

2.3 Optional requirements

Although the requirements in this Section are optional they will make the training session a bit more fun and easier.

- pgAdmin:You can run queries to the database by installing pgAdmin, an application that will serve as a web interface and will make things easier. To install pgAdmin, type:

Listing 18: Install pgAdmin4

```
    sudo apt install pgadmin4
```

- QGIS Desktop: Finally, in order to visualize on the map the results obtained when we run the queries it is recommended to install the latest version of QGIS Desktop with the OpenStreetMap layers for QGIS.

Attribute	Description
Timestamp	the time at which the message was received (EET)
Ship ID	unique identifier for each ship
Longitude	the longitude of the current ship position
Latitude	the latitude of the current ship position
Heading	heading of the ship's bow in degrees (0 corresponds to north)
Course Over Ground (COG)	heading the vessel travels to in degrees (0 corresponds to north)
Speed Over Ground (SOG)	Speed in knots
Ship Name	the name of the ship
Ship Type	AIS reported ship-type
Draught	the draft of the ship (0.1 to 25.5 metres)
Size Bow	Distance from bow to the AIS transponder on board the ship
Size Stern	Distance from stern to the AIS transponder on board the ship
Size Port	Distance from port side to the AIS transponder on board the ship
Size Starboard	Distance from starboard side to the AIS transponder on board the ship
Destination	AIS reported destination

Table 1: Dataset attributes

3 AIS Protocol and Dataset

AIS stands for Automatic Identification System and is a global tracking system that allows vessels to be aware of vessel traffic in their vicinity and to be seen by that traffic. Through this tracking system vessels broadcast information about their location (i.e., GPS coordinates) and behaviour (e.g., speed, course, etc.), as well as information about their characteristics such as vessel size, draught and destination. All vessels over 300 gross tonnage are obliged by the International Maritime Organization (IMO) to carry an Automatic Identification System (AIS) transponder on board.

Dataset: During the training session, we will be using a dataset that contains AIS messages collected from a Terrestrial AIS receiver (T-AIS) installed on top of the building of the Department of Informatics and Telematics of Harokopio University that was provided by MarineTraffic. The dataset covers the Saronic Gulf (Greece) during a two-week period starting at March 10th, 2020 and ending at March 19th, 2020. The dataset provides information for 773 unique vessels and contains 3,543,482 AIS records in total each comprising 15 attributes as described in Table 1.

The dataset can be found here. To download the dataset through the command line type:

Listing 19: Download dataset

```
sudo apt install python-pip
pip install gdown
gdown https://drive.google.com/uc?id=1F0yKCOfo7iIgWbkuReytCxrnD3vz1GAS
```

4 Managing Ship Trajectories

4.1 Preparing the Database

Create a new database called hua_ais:

Listing 20: Create Database

```
CREATE DATABASE hua_ais
```

If you are using the command line connect to the database by typing:

Listing 21: Connect to the Database

```
\c hua_ais
```

If you are using pgAdmin, simply open the query tool inside the newly created database. Then use your SQL editor to create the MobilityDB extension as follows:

Listing 22: MobilityDB extension

```
CREATE EXTENSION MobilityDB CASCADE;
```

The CASCADE command will additionally create the PostGIS extension. Now create a table in which the CSV file will be loaded:

Listing 23: Create AIS table

```
CREATE TABLE AISInput(
T        timestamp,
ID varchar(100),
Longitude float,
Latitude float,
Heading integer,
COG float,
SOG float,
ShipName varchar(100),
ShipType varchar(100),
Draught float,
SizeBow float,
SizeStern float,
SizePort float,
SizeStarboard float,
Destination varchar(100),
Geom geometry(Point, 4326)
);
```

4.2 Loading the data

To import a CSV file data into a PostgreSQL database we can use the COPY command as follows:

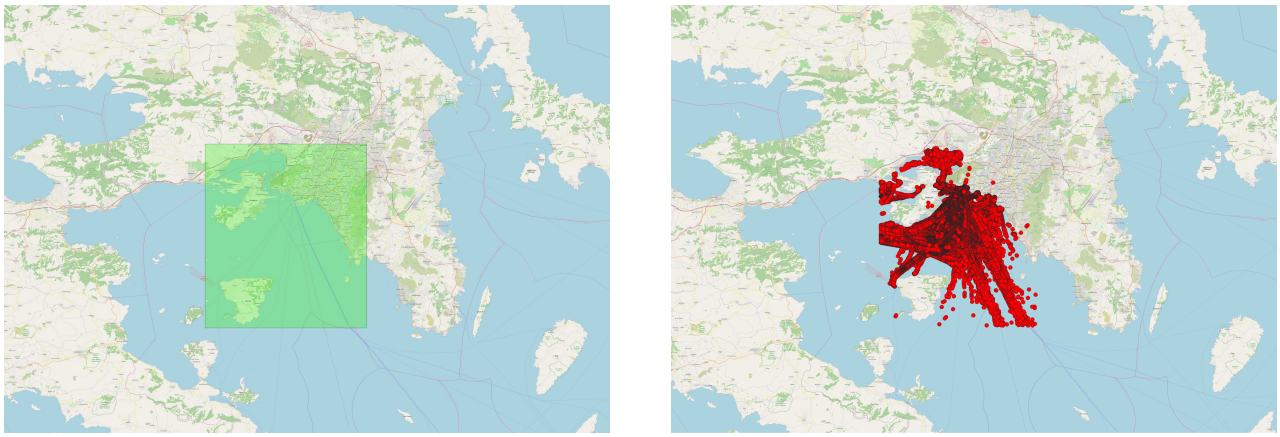
Listing 24: Load data into table

```
COPY AISInput(T, ID, Longitude, Latitude, Heading, COG, SOG, ShipName, ShipType,\
Draught, SizeBow, SizeStern, SizePort, SizeStarboard, Destination)
FROM '/home/test/workshop_hua_ais.csv' DELIMITER ',' CSV HEADER;
```

Next, we create the spatial points for each AIS message:

Listing 25: Create spatial points

```
UPDATE AISInput SET
Geom = ST_SetSRID( ST_MakePoint( Longitude, Latitude ), 4326);
```



(a) Bounding box of the filtered dataset.

(b) The remaining AIS positions after the filtering.

Figure 1: Area under surveillance.

Now, we do some basic cleaning of the dataset. First, we filter out points that are outside of the bounding box as shown in Figure 1a:

Listing 26: Filter data

```
CREATE TABLE AISInputFiltered AS
SELECT DISTINCT ON(ID,T) *
FROM AISInput
WHERE Longitude BETWEEN 23.378894 and 23.815601 AND
Latitude BETWEEN 37.664548 AND 38.057034;
```

A tool with which you can extract a bounding box can be found here.

To export the data of the "AISInputFiltered" table to a csv file type in the SQL editor the following:

Listing 27: Export to CSV

```
COPY
AISInputFiltered(t,ID,Longitude,Latitude,Heading,COG,SOG,ShipName,ShipType,Draught,
SizeBow,SizeStern,SizePort,SizeStarboard,Destination)
TO '/home/test/filtered_ais.csv' WITH (FORMAT CSV, HEADER);
```

A visual illustration of the filtered dataset can be seen in Figure 1b

4.3 Constructing trajectories

In this step, we construct trips or trajectories from AIS messages, a spatiotemporal data type that describes the movement of the vessel in space and in time. MobilityDB builds on the coordinate transformation feature of PostGIS.

Listing 28: Construction of trajectories

```
CREATE TABLE Trips(ID, Trip) AS
SELECT ID,
tgeompointseq(array_agg(tgeompointinst( ST_Transform(Geom, 4326), T) ORDER BY T))
FROM AISInputFiltered
GROUP BY ID;
```

In order to visualize the trips, we extract the spatial aspect of the trip and update the table:

Listing 29: Create table of trips

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

We then store the trips to csv:

Listing 30: Extract trips to csv

```
COPY (SELECT ST_AsText(traj) FROM trips)
TO '/home/test/ws/trips.csv' WITH (FORMAT CSV, HEADER);
```

The visualization of the trajectories can be seen in Figure 2.

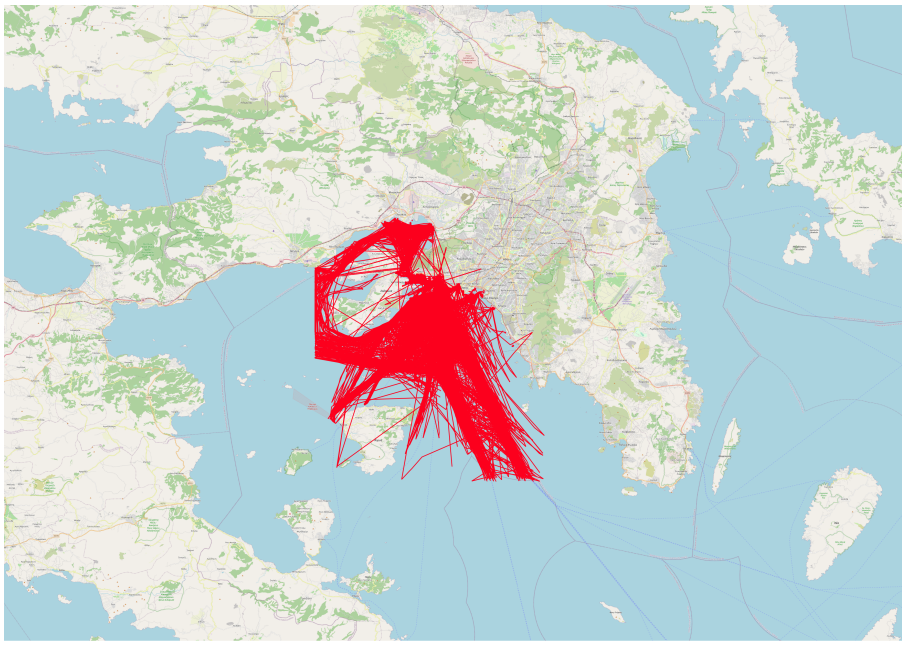
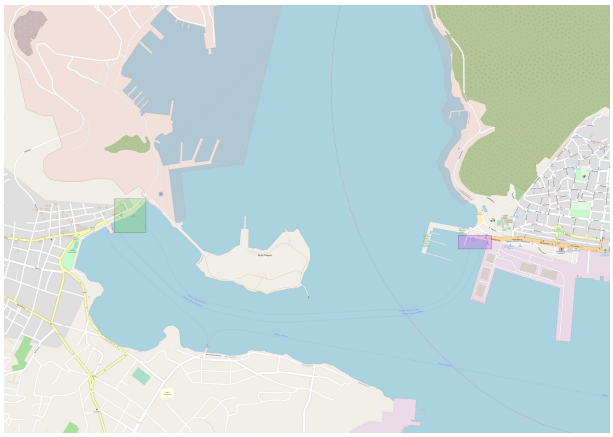
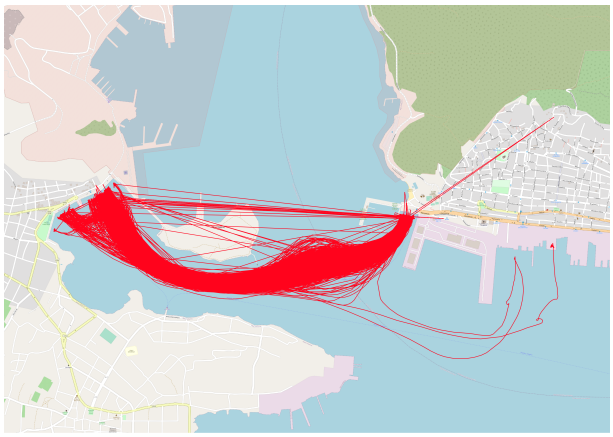


Figure 2: The linestrings of the trajectories.



(a) The ports of Perama and Paloukia in the right and left side of the Figure respectively.



(b) The trips between Athens and the island of Salamina.

Figure 3: Vessels that travel between two ports.

4.4 Analyzing trajectories

In this section, we dive a bit more in the features and functions of the MobilityDB. In the dataset, there are several vessels that travel between Athens (Perama port) and the island of Salamina (Paloukia port). The two ports can be seen in Figure 3a. The goal is to simply identify which ships do so, and to count how many two-way trips they did in the dataset. This can be expressed by the following query:

Listing 31: Extract trips between the two ports

```
CREATE INDEX Trips_Trip_Idx ON Trips USING GiST(Trip);

WITH Ports(Paloukia, Perama) AS (
SELECT ST_MakeEnvelope(23.528269227, 37.9644819698,
                       23.5306885795, 37.9665246767, 4326),
       ST_MakeEnvelope(23.5546458997, 37.9634903768,
                       23.557177905, 37.964298173, 4326)
)
SELECT T.*, Paloukia, Perama
FROM Ports P, Trips T
WHERE intersects(T.Trip, P.Paloukia) AND intersects(T.Trip, P.Perama);
```

This query creates two envelope geometries that represent the locations of the two ports, then intersects them with the spatiotemporal trajectories of the ships. The "intersects" function checks whether a temporal point has ever intersected a geometry. To speed up the query, a spatiotemporal GiST index is first built on the Trip attribute. The trips between Perama and Paloukia are illustrated in Figure 3b. To count how many two-way trips each vessel did, we extend the previous query as follows:

Listing 32: Count how many two-way trips each vessel did

```
WITH Ports(Paloukia, Perama) AS (
SELECT ST_MakeEnvelope(23.528269227, 37.9644819698,
                       23.5306885795, 37.9665246767, 4326),
       ST_MakeEnvelope(23.5546458997, 37.9634903768,
                       23.557177905, 37.964298173, 4326)
)
SELECT ID, (numSequences(atGeometry(T.Trip, P.Paloukia)) +
numSequences(atGeometry(T.Trip, P.Perama))) AS NumTrips
FROM Ports P, Trips T
WHERE intersects(T.Trip, P.Paloukia) AND intersects(T.Trip, P.Perama)
```

The function "atGeometry" restricts the temporal point to the parts where they are inside the given geometry. The result is thus a temporal point that consists of multiple pieces (sequences), with temporal gaps in between. The function "numSequences" counts the number of these pieces. The results of the query can be seen in Figure 4.

	id character varying (100)	numtrips integer
1	CAAA18900011505426	77
2	HDHD41882392206695	64
3	BKZT14731869247997	99
4	BLHU54428683350633	73
5	CRUC86263254975353	89
6	DOJU11473329264852	69
7	GPSC39583190574178	103
8	IXFR19391976554025	121
9	KQII99826024067145	104
10	LHDM47738578379828	95
11	PGBG71292303044820	77
12	PKYS63382368960885	94
13	PNYA74056376535996	93
14	PRXG78101660586023	37
15	RRTT08292434805330	75
16	TERT47638213967652	111
17	WBBD46513444533339	29
18	WIVS56782319755913	104
19	YXWO17211581103136	82
20	ZBYS26611448915120	85

Figure 4: The number of two-way trips per vessel between Athens and the island of Salamina.

As already mentioned, the dataset covers a period starting at March 10$^{\text{th}}$, 2020 and ending at March 19$^{\text{th}}$, 2020. The period was selected to contain vessel trajectories before and after the corona virus lockdown which started at March 16$^{\text{th}}$. Therefore, we present queries that are a bit more interesting. The following two queries count the total number of active trips between two different periods. The first period is $p_1 \in [2020/03/12, 2020/03/15]$ and the second period is $p_2 \in [2020/03/16, 2020/03/19]$, before and after the lockdown respectively. After running the queries we can observe that the number of total trips were reduced from 1,677 to 1,574 which shows that only a few days after the lockdown the maritime domain has been affected.

Listing 33: Count the total number of trips that were active 4 days before the corona virus lockdown

```
WITH TimeSplit(Period) AS (
SELECT period(D, D + interval '1 day')
FROM generate_series(timestamptz '2020-03-12 00:00:00',
        timestamptz '2020-03-15 23:00:00', interval '1 day') AS D )
SELECT COUNT(*)
FROM TimeSplit S, Trips T
WHERE S.Period && T.Trip AND atPeriod(Trip, Period) IS NOT NULL;
```

Listing 34: Count the total number of trips that were active 4 days after the corona virus lockdown

```
WITH TimeSplit(Period) AS (
SELECT period(D, D + interval '1 day')
FROM generate_series(timestamptz '2020-03-16 00:00:00',
        timestamptz '2020-03-19 23:00:00', interval '1 day') AS D )
SELECT COUNT(*)
FROM TimeSplit S, Trips T
WHERE S.Period && T.Trip AND atPeriod(Trip, Period) IS NOT NULL;
```

The queries split the period into smaller one-day periods and for each sub-period the number of trips is counted. Finally, the number of trips per day are aggregated to show the total number of active trips during the period. To count the number of total trips during a period, we can use a simpler query, but the query was made more complex to show more functionalities of the MobilityDB.

References

[1] Mohamed S. Bakli, Mahmoud Attia Sakr, and Esteban Zimányi. Distributed moving object data management in mobilitydb. In Varun Chandola, Ranga Raju Vatsavai, and Ashwin Shashidharan, editors, *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial@SIGSPATIAL 2019, Chicago, IL, USA, November 5th, 2019*, pages 1:1–1:10. ACM, 2019.

[2] Mohamed S. Bakli, Mahmoud Attia Sakr, and Esteban Zimányi. Distributed mobility data management in mobilitydb. In *21st IEEE International Conference on Mobile Data Management, MDM 2020, Versailles, France, June 30 - July 3, 2020*, pages 238–239. IEEE, 2020.

[3] Mohamed S. Bakli, Mahmoud Attia Sakr, and Esteban Zimányi. Distributed spatiotemporal trajectory query processing in SQL. In Chang-Tien Lu, Fusheng Wang, Goce Trajcevski, Yan Huang, Shawn D. Newsam, and Li Xiong, editors, *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020*, pages 87–98. ACM, 2020.

[4] Esteban Zimányi, Mahmoud Attia Sakr, Arthur Lesuisse, and Mohamed S. Bakli. Mobilitydb: A mainstream moving object database system. In Walid G. Aref, Michela Bertolotto, Panagiotis Bouros, Christian S. Jensen, Ahmed Mahmood, Kjetil Nørvåg, Dimitris Sacharidis, and Mohamed Sarwat, editors, *Proceedings of the 16th International Symposium on Spatial and Temporal Databases, SSTD 2019, Vienna, Austria, August 19-21, 2019*, pages 206–209. ACM, 2019.