Maritime Data Processing in Relational Databases

Laurent Etienne, Cyril Ray, Elena Camossi, Clément Iphar

Maritime Data Processing in Relational Databases

Laurent Etienne, Cyril Ray, Elena Camossi, Clément Iphar

Abstract Maritime data processing research has long used spatio-temporal relational databases. This model suits well the requirements of off-line applications dealing with average-size and known in advance geographic data that can be represented in tabular form. This chapter explores off-line maritime data processing in such relational databases and provides a step-by-step guide to build a maritime database for investigating maritime traffic and vessel behaviour. Along the chapter, examples and exercises are proposed to build a maritime database using the data available in the open, *heterogeneous, integrated dataset for maritime intelligence, surveillance, and reconnaissance* that is described in [41]. The dataset exemplifies the variety of data that are nowadays available for monitoring the activities at sea, mainly the Automatic Identification System (AIS), which is openly broadcast and provides worldwide information on the maritime traffic. All the examples and the exercises refer to the syntax of the widespread relational database management system *PostgreSQL* and its spatial extension *PostGIS*, which are an established and standard-based combination for spatial data representation and querying. Along the chapter, the reader is guided to experience the spatio-temporal features offered by the database management system, including spatial and temporal data types, indexes, queries and functions, to incrementally investigate vessel behaviours and the resulting maritime traffic.

Laurent ETIENNE
LabISEN, FRANCE, e-mail: laurent.etienne@isen-ouest.yncrea.fr

Cyril RAY
Naval Academy Research Institute, FRANCE, e-mail: cyril.ray@ecole-navale.fr

Elena CAMOSSI
NATO STO Centre for Maritime Research and Experimentation, ITALY, e-mail: elena.camossi@cmre.nato.int

Clément IPHAR
NATO STO Centre for Maritime Research and Experimentation, ITALY, e-mail: clement.iphar@cmre.nato.int

1 Introduction

The analysis and the processing of data from automated surveillance technologies like the Automatic Identification System (AIS), Synthetic Aperture Radar (SAR), satellite images and coastal radars (*cf.* the chapter of Bereta et al. [7]) is an emerging topic. Once fused with contextual information like coastlines, nautical charts, fishing areas, maritime protected areas, sea state and weather conditions, these positioning data can be analysed in order to understand vessel mobility, monitor maritime traffic, uncover activities or risks for the environment, living resources and navigation, and so on [41]. The research is expeditiously progressing, and many analysis techniques are applied to attain the above major goals using maritime navigational data. Some of the leading maritime informatics research topics include:

- *Vessel trajectory analysis and prediction*, recently focusing mainly on AIS and complementary sources to predict and analyse vessel trajectories [8].
- *Traffic forecasting*, like port volume handling and cargo throughput forecast [24].
- *Collision prevention*, analysing risk of ships collisions through the analysis of their navigational behaviour [22].
- *Ship detection, classification and identification*, analysing videos from cameras in delimited areas, for instance to detect small vessels [25], or remote sensing images in larger areas [14].
- *Anomaly detection*, mostly adopting data-driven methods to model normal traffic against which any irregular behaviour is associated to potential threats [44].
- *Analysis of human activities at sea*, such as fishing, illegal traffic (human beings, narcotics, goods), piracy [39].
- *Multi-source information fusion*, to reduce uncertainty in the data available and achieve better accuracy in analysis [33].
- *Dynamics of maritime transportation networks*, where trajectory data are considered at an aggregated level through network abstractions designed to analyse behavioral patterns [60].

Research work processing maritime data requires modelling and analysing moving objects associated to maritime navigation and traffic. Considering an existing batch of data, two main processing approaches exist. Some works consider direct processing of data files, using general-purpose data science environments (e.g. *Python*, *R*, *Matlab*) or developing customised tools for spatio-temporal data processing (e.g. [3]). Other works use database management systems (DBMS), which nowadays offer a native support for modelling spatial information, optimised functionalities for spatial indexing and analysis, and efficient integration with the query language.

The latter approach, which is the one illustrated in this chapter, enables a quick development of moving object applications. Thanks to built-in spatial data types and extended DBMS support, developers can exploit the spatial dimension of the application entities. For instance, they can explicitly represent spatial objects and execute spatial queries to compare their topological properties or to calculate their geographical distance. The available spatial types of such DBMS can be extended

with user-defined data types and functions, tailored to the specific application or task. The soundness and the efficiency of the resulting model are guaranteed by the DBMS.

This chapter focuses on the support offered by relational DBMS, which are proposed for maritime data storage and querying. While time-consuming for handling very large datasets, and while being mainly used for off-line analysis, relational DBMS suit well the requirements of applications dealing with average-size heterogeneous data that can be described according to a known structured and fixed schema, i.e. where data properties are known in advance and can be represented in tabular form.

Along the chapter, many SQL (Structured Query Language) [35] examples are presented to the reader, who is asked to solve some exercises.[1] A basic knowledge of relational databases and SQL is necessary to understand the examples and solve the exercises. All together, examples and exercises illustrate the spatio-temporal capabilities of DBMSs that can be progressively combined to analyse maritime data for investigating vessel traffic. The examples in this chapter refer to *PostgreSQL*,[2] a widespread relational DBMS. *PostgreSQL* is a very popular DBMS, fully open source, very robust, and its spatial support, which is provided by *PostGIS* [53], is compliant to well established standards for representing spatial features. *PostgreSQL* also integrates a native temporal support, which is necessary to analyse vessel movements, that corresponds to Allens algebra operators [1].

The rest of the chapter is organised as follows. In Section 2 we overview the existing approaches for representing and managing spatial data and moving objects. Section 3 introduces the software the reader needs in order to create, query and visualise a maritime database. Section 4 explains the steps required to create a maritime database, given the *heterogeneous, integrated dataset for maritime intelligence, surveillance, and reconnaissance* that is described in [41]. Using this maritime dataset, section 5 illustrates typical queries and functions, useful to derive additional information and reason on vessel movements. Finally, Section 6 concludes the chapter.

2 Systems for spatial data and moving objects

In this section, we overview how spatial, specifically geographic data, and moving objects are handled, either using Geographic Information Systems (GIS), or DBMS. The approaches and software products supporting only spatial data are introduced first, followed by some of the existing systems for spatio-temporal data.

---

[1] Exercise solutions are available, with additional material (e.g. resulting geographic data), online [*NEW DOI OBJECT TO CREATE ON ZENODO*], together with the reference dataset used in the examples.

[2] `https://www.postgresql.org`

2.1 Handling spatial data

For decades, the representation and analysis of geographical information have been investigated in the area of Geographic Information Systems (GIS). GIS, like *ArcGIS*[3], *GRASS*[4], and *QGIS*[5] naturally evolved from the digitalisation of cartographic maps. GIS have efficient geographic visualisation and rendering capabilities, which can be combined with multi-layer spatial analysis functions like distance/buffer and topological queries, overlay operations, surface and terrain analysis. However, traditional GIS alone cannot deal with the increasing quantity of data, and do not suite the semantically-driven integration of heterogeneous information that involves both spatial and non-spatial attributes, or complex knowledge discovery tasks like supervised classification. All these tasks require an integrated representation of geographic and descriptive data features, and a greater flexibility in low-level data manipulation than the one offered by traditional GIS, which keep the management of geospatial data and other data types separated [43].

To address the needs of emerging spatial-driven applications, relational database systems, once geared towards providing efficient support for simple objects with discrete attributes, have been extended to represent and query geographic data in a natural way. For instance, *Oracle*[6] and *Postgres* (now *PostgreSQL*), were expanded with spatial modules, respectively *Oracle Spatial* [26] (now fully integrated in the main DBMS) and *PostGIS*. Nowadays, all the main DBMS, including *Microsoft SQL*[7], *MySQL*[8], *SQLite*[9] with *SpatiaLite*, provide support to cope with geographic data.

The data model of these DBMS has been extended with data types and structures for managing geometric data, relying on established international standards and recommendations[10,11] Spatial operations and types are fully integrated with the query language, and the DBMS engine is enhanced to map from the query language to the spatial features. Also, spatial indexing is provided for query optimisation. Most of the products include *B-Tree+* and *R-Tree* [17] indexes. *PostgreSQL* additionally implements the *Generalized Search Tree (GiST)* [18] to develop customised indexing, mixing, for instance, spatial and non-spatial optimisation.

---

[3] `https://arcgis.com`

[4] `https://grass.osgeo.org`

[5] `https://qgis.org`

[6] `https://www.oracle.com`

[7] `https://www.microsoft.com/en-us/sql-server/sql-server-2017`

[8] `https://www.mysql.com`

[9] `https://sqlite.org`

[10] Open Geospatial Consortium (OGC) Simple Features Specification for SQL: Simple Feature Access - Part 1: Common Architecture `https://www.opengeospatial.org/standards/sfa`; Simple Feature Access - Part 2: SQL Option `https://www.opengeospatial.org/standards/sfs`

[11] International Standard Organization's (ISO) ISO19107:2003 Geographic InformationSpatial Schema `https://www.iso.org/standard/26012.html`

Most DBMS spatial extensions provide mature support for geometric data. Some products include also extensions for raster data (e.g. *Oracle Raster* and *pgraster*, which is integrated in *PostGIS*). Also, few products support 3D representations, including *PostGIS*.

This support can be further enhanced with novel abstract data types to represent spatial (and spatio-temporal) data, tailored to the application, analogously to *object-oriented DBMS* (OODBMS). These (object-) relational DBMS, taking advantage of both spatial and object-oriented functionalities, have been established as main players on the market for storing and querying spatial data. For this reason, nowadays most of GIS enable the use of a spatial DBMS as data storage layer. This hybrid solution combines the advantages of the efficient front-end of GIS and the robust and efficient storage and retrieval capabilities of spatial DBMS.

Few NoSQL, or *Not-only-SQL*, DBMS, i.e. all the DBMS that adopt alternative models to the (object-)relational one, offer spatial extensions. Among them, few research proposals exist to extend OODBMS to support spatial features, leveraging the model extensibility as described above. More recent NoSQL DBMS, like *MongoDB* and *DocumentDB*, which adopt a semi-structured document-based data model, enable the definition of spatial entities supporting *GeoJSON* objects and geo-spatial queries. *ArangoDB* is a multi-model (key value, graph and document) NoSQL DBMS that supports natively geo-spatial querying and indexing. *Rasdaman* (which stays for *raster data manager*) is a multi-dimensional DBMS for scientific data, which can be queried with an SQL-like language.

NoSQL DBMS relax some data modelling constraints to handle unstructured or semi-structured data, and improve *scalability* performance as required for big data and parallel processing. For instance, in order to process in parallel multiple vessel trajectories, and multiple areas, they enable to increase, seamlessly for the application, the number of nodes in the infrastructure that are used to host the database, and maintain replicas of the data. However, the gain in efficiency and flexibility comes at the price of relaxing some of the properties that, in traditional SQL databases, guarantee data consistency (namely, ACID properties: i.e. atomicity, consistency, isolation, and durability).

It should be noted that NoSQL solutions do not always outperform relational DBMS. For instance, the authors in [32] compare *PostgreSQL/PostGIS* and *MongoDB* performance on a maritime dataset of vessel positions, and show that *PostgreSQL/PostGIS* outperforms *MongoDB*. As shown in the *Knowledge base of relational and NoSQL DBMS*,[12] relational DBMS remain amongst the most popular DBMS.

[12] `https://db-engines.com/en/ranking`

2.2 Spatial-temporal and movement data in databases

While widely studied, the representation of the temporal dimension of spatio-temporal and moving object data in databases remains rather limited or in the research phase [48]. In [19] the authors illustrate a spatio-temporal extension of the Object Data Management Group (*ODMG*) model, which is a *de facto* standard model for OODBMS, and of its query language, *OQL*. A renowned spatio-temporal model is the *SECONDO*'s one, which exploits abstract data types to model moving objects [57] and spatio-temporal operations among them.

PostgreSQL natively integrates a support for temporal data that nicely combines with the spatial extension of *PostGIS*. Additionally, the reader may consider two groups of alternatives. As a first option, a time-series DBMS can be used. For instance, *TimescaleDB*[13] is an open-source time-series DBMS developed as *PostgreSQL*'s extension, fully SQL and *PostgreSQL* compliant, which can integrate *PostGIS* to efficiently handle spatio-temporal time-series data like Internet-of-Things observations. Given an SQL table with temporal and location attributes, it transforms it and partitions the data according to the temporal and spatial dimensions and adds indexes for improving access performance. It also offers a small set of analytic functions (e.g. *first*, which provides the first value of time ordered series) that can be used jointly with *PostGIS*'s functions.

The second group of options implements a full spatio-temporal support. For instance, *PostGIS-T* extends *PostgreSQL* on the basis of a formal spatio-temporal algebra [50]. Analogously, *Pg-Trajectory* [28] which is developed by the Data Mining Lab of the Georgia State University, is a *PostgreSQL*/*PostGIS* extension designed for spatio-temporal data.

Other spatio-temporal proposals, still based on *PostgreSQL*, are *Hermes* and *MobilityDB*. *Hermes* [38] is an in-DBMS framework for data mining, more particularly for the processing of moving objects. It is a *Python* library, which is event-driven, failure-handling and *PostgreSQL*-talking, and enables an easy implementation of Python processes that require *PostgreSQL* communication. *MobilityDB* [55] is a *PostgreSQL* extension for mobility data management, which follows the OGC Moving Features Access specification[14] and defines the operations applicable to time-varying geometries.

3 Building a maritime information system

A Maritime Information System can be defined as a system for the collection, processing, storage and delivering of maritime information through a visualisation interface. The main purpose of a DBMS, when integrated to such a Maritime Informa-

[13] TimescaleDB `https://www.timescale.com`

[14] OGC Moving Features Access `http://www.opengis.net/doc/is/movingfeatures-access/1.0`

tion System, is to organise, store, access and process the maritime data. The database can be handled and queried through a management tool or an external programming environment (e.g. *Python*, *R*, *Java*). In order to visualise the data, dedicated applications and Application Programming Interface (API), e.g. Grafana,[15] Data Driven Documents (D3),[16] Kibana,[17] visual analytics tools (*cf.* the chapter of Andrienko et al. [2]) or a GIS software like QGIS can be used.

Action required

Download and install the following software to prepare the working environment (let us note that *PostGIS* and *pgAdmin* are in general part of the last *PostgreSQL* bundles):

- *PostgreSQL*: `https://www.postgresql.org`;
- *PostGIS*: `https://postgis.net`;
- *pgAdmin*: `https://www.pgadmin.org`;
- *QGIS*: `https://www.qgis.org`.

3.1 PostgreSQL ***DBMS***

The DBMS employed in this chapter to illustrate the use of relational databases for maritime data is *PostgreSQL*. *PostgreSQL* is an open source, standard compliant and robust DBMS. It adopts and extends the *Structured Query Language (SQL)* [35] for data manipulation and querying. *PostgreSQL* runs on all major operating systems and can handle big datasets. *PostgreSQL* is highly extensible: the users can define their own data types, build customised new functions and interact with the DBMS through the query language *SQL*, using different programming languages like *C*, *Perl*, *Java*, *Python*, *R*, *JavaScript*, etc., and from shell scripts.

3.2 PostGIS ***spatial extension***

The spatial component of the maritime dataset plays a fundamental role for maritime situational awareness. *PostGIS* is a powerful add-on to *PostgreSQL* that adds geospatial capabilities to the DBMS. *PostGIS* integrates both raster and vector types of data which are fully compliant with the *OGC Simple Features Specification for SQL*, and a large number of spatial operations. *PostGIS* also implements multiple spatial indexing methods, including *R-Tree* and *GiST*. Although *PostgreSQL* and *PostGIS* are usually used with two-dimension (2-D) geometries (coordinates *X* and *Y*), *PostGIS* supports the addition of a third dimension (*Z*), allowing to reason with three-dimensional geometries.

[15] Grafana `https://grafana.com`

[16] Data Driven Documents (D3) `https://d3js.org`

[17] Kibana `https://www.elastic.co/products/kibana`

3.3 PGAdmin management tool

Once the DBMS is installed, management tools are useful to setup and query the database. *pgAdmin* is a multi-platform software that is dedicated to manipulate *PostgreSQL* databases. Once connected to a *PostgreSQL* cluster (i.e. a collection of *PostgreSQL* servers), *pgAdmin* can create and query new databases and tables. Its powerful query tool supports colour syntax highlighting and graphical query plan displaying.

3.4 QGIS visualization tool

QGIS is the last component of our Maritime Information System. It is a user-friendly open-source GIS, which can retrieve, manage, display and analyse the geographic data stored in the *PostgreSQL*/*PostGIS* database. In this chapter, *QGIS* will be used to exemplify the visualisation and creation of maps to show the results of the spatial queries (see Figure 1). Interested readers can refer to Anita Graser's blog, "Free and Open Source GIS Ramblings",[18] for practical illustrations on advanced functionalities for movement data. Moreover, the readers can refer to the *QGIS* online user guide[19] to get started with *QGIS*.

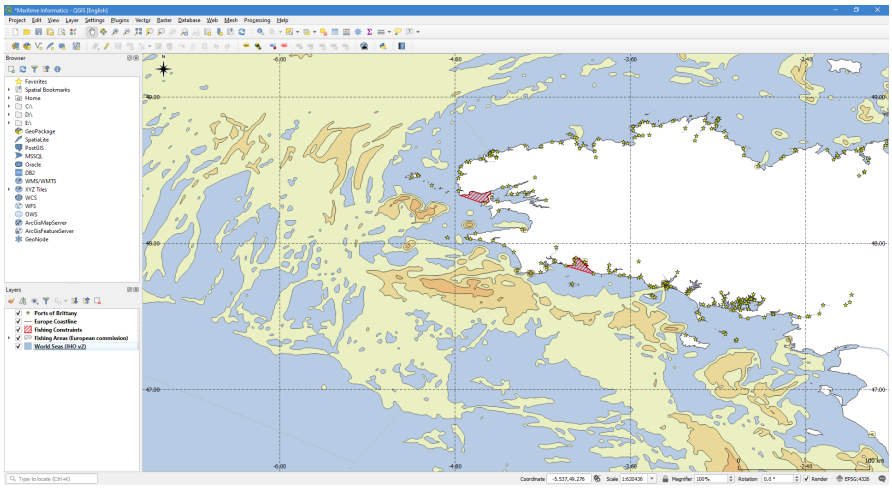

Fig. 1 The *QGIS* desktop visualising five geographical layers (all available in the dataset described in [41]): fishing areas (by [56]), ports of Brittany, Europe coastline, fishing constraints and world seas.

[18] "Free and Open Source GIS Ramblings" `https://anitagraser.com/`

[19] `https://docs.qgis.org/3.4/en/docs/user_manual/`

3.5 Getting the maritime dataset

The SQL examples of this chapter are based on a freely available open dataset that has been processed using *PostgreSQL* and its spatial extension *PostGIS*. The dataset is heterogeneous, and contains four categories of data:

- *Navigation data*, i.e. historical AIS positions of vessels navigating around a major shipping route;
- *Vessel-oriented data* (e.g. ship registers, i.e. lists of vessels with their characteristics);
- *Geographic data* (i.e. cartographic, topographic and regulatory context of vessel navigation);
- *Environmental data* (i.e. climatic and sea-state related information).

All data are temporally and spatially aligned to allow for efficient and advanced spatio-temporal analysis. The dataset covers a time period of six months (i.e. from October 1st 2015 to March 31st 2016) and provides around 18.6 million ship positions collected from 4,842 ships over the Celtic sea, the North Atlantic ocean, the English Channel, and the Bay of Biscay in France. For additional details on the dataset, see [41].

Action required

Access the dataset at: `https://zenodo.org/record/1167595`

The AIS data included in the dataset aggregate different message types, and are stored in two separated files, which contain respectively the positioning messages of ships, and their nominative messages. Positioning messages are AIS messages of type ITU 1, ITU 2, ITU 3, ITU 18, and ITU 19, where ITU stands for International Telecommunication Union. Nominative messages are AIS messages of type ITU 5, ITU 19, and ITU 24, which contain static information about the emitting vessel, such as its name, dimensions or destination, identifiers like the International Maritime Organization - IMO - number and the Maritime Mobile Service Identity - MMSI - code. Two additional files, containing respectively search and rescue messages, i.e. ITU 9, and aids to navigation messages, i.e. ITU 21, have been also included.

Remark ! SQL queries

The SQL queries presented in this chapter focus on typical and useful features for maritime informatics. They complement the integration queries available in folder "[Q1] Integration Queries" of the maritime dataset. All the queries presented in this chapter have been prepared using local installations (on Windows x86-64) of *PostgreSQL* 11.5, *PostGIS* 2.5.3, and *pgAdmin*4 v4.18.

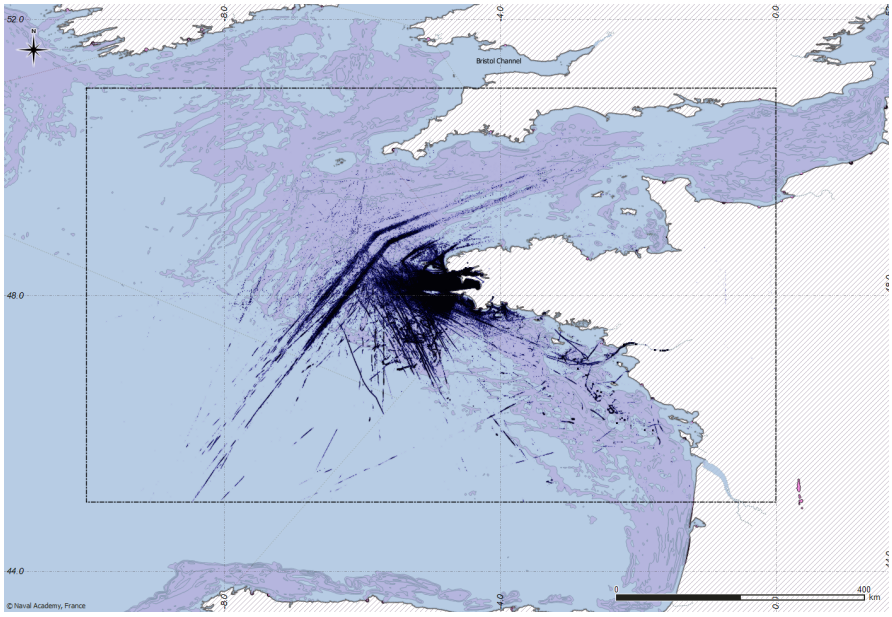

Fig. 2 AIS navigation data in the open dataset of [41]. The purple polygons represent fishing zones computed from navigation data by Vespe et al. [56].

4 Creation of the Maritime Database

In this section, a new maritime database will be created and populated. Assuming that the software packages described in Section 3 have been installed, it is possible to connect to the *PostgreSQL* database server from *pgAdmin* interface, using the credentials provided during the installation. Once a connection to the database is established, the following command can be executed (in menu *Tools* → *Query Tool*) to create a new empty database called `maritime_informatics`.

SQL *Q4.0.1*

```
CREATE DATABASE "maritime_informatics";
```

Remark ! Important note about literals

PostgreSQL automatically converts to lower case all tables and fields names. If you want to force the use of capitals in tables or fields names, you must quote them (i.e. ``tableName'' or ``fieldName''). To distinguish field names and strings in queries, *PostgreSQL* uses double quotes (`` and '') for field names and single quotes (` and ') for string values.

4.1 The database schema and the tables

A database is organised and structured using one or more *database schemas* and several *tables*. Each schema defines a workspace where new data types, functions and operators can be defined. Organising a database using different schemas is useful for various reasons:

- To organise database objects within logical groups and to ease their management;
- To isolate third-party applications and allow for duplicate objects or function names in different schemas;
- To grant users (and groups of users) appropriate access to data and functions that belong to different schemas.

Schema and tables for AIS data

AIS transceivers on board of vessels broadcast vessel positions and interesting information about static ship characteristics such as the vessel name, *callsign* (i.e. unique vessel identifier for radio broadcasts), IMO number and ship type. Each AIS transceiver has its own Maritime Mobile Service Identity (MMSI), which is registered and attributed by country of flag. This number identifies the source of the AIS message (`sourcemmsi`). Usually, AIS transceivers broadcast this information to all the ships in the vicinity. However, some AIS messages can be addressed to one specific AIS transceiver, identified using its MMSI (`destinationmmsi`). Ship's tenders can also have their own AIS transceiver. The tender's transceiver reports in the AIS messages the MMSI of the ship it belongs to (`mothership`).

For the examples presented in this chapter, AIS information will be organised within a schema named `ais_data`. The SQL instructions below show how to create the schema `ais_data` into the database and create, inside this schema, a new table `static_ships` to store voyage and nominative pieces of information. Note that each attribute has a specific type (`integer`, `text`, `double`, etc.).

SQL **Q4.1.1**

```
CREATE SCHEMA "ais_data";

CREATE TABLE ais_data.static_ships(
  id bigserial, -- unique identifier of the row in the table
  sourcemmsi integer, -- MMSI identifier of the source ship's transceiver
  imo integer, -- IMO number of the ship, linked to the vessel structure
  callsign text, -- Callsign of the ship
  shipname text, -- Ship name
  shiptype integer, -- Type of the vessel, according to AIS specifications
  to_bow integer, -- Distance of the AIS transceiver from bow (front) of
     vessel, rounded to the nearest meter
  to_stern integer, -- Distance of AIS antenna form stern (back)
  to_starboard integer, -- Distance of AIS antenna form starboard (right)
  to_port integer, -- Distance of AIS antenna form port (left)
  eta text, -- Estimated Time of Arrival to destination
  draught double precision, -- Ship draught
  destination text, -- Declared ship destination
  mothershipmmsi integer, -- MMSI of the mothership
  ts bigint, -- Timestamp of the AIS frame
  CONSTRAINT static_ships_pkey PRIMARY KEY (id) -- Primary Key
);
```

4.2 Loading the data into the database

The new table can now be populated. Data can be manually inserted into the existing tables using the SQL statement `INSERT INTO` or, alternatively, directly from files using the *PostgreSQL* command `COPY FROM`. The following example shows how to load into the table `static_ship` the static ship information stored in the Comma Separated Values (CSV) file `nari_static`, which is included in the maritime dataset (in folder "[P1] AIS Data"). In the command below, `<path_to_dataset>` must be replaced with the local path to the file.

PostgreSQL **Q4.2.1**

```
COPY ais_data.static_ships
     (sourcemmsi,imo,callsign,shipname,shiptype,to_bow,to_stern,to_starboard,
      to_port,eta,draught,destination,mothershipmmsi,ts)
FROM '<path_to_dataset>/[P1] AIS Data/nari_static.csv',
DELIMITER ',' CSV HEADER;
```

As soon as the database table is filled with *static data*, it can be queried using SQL commands. The query in the next example counts how many different ship names are listed in the table `static_ships` ([Answer: 4,824]).

SQL **Q4.2.2**

```
SELECT DISTINCT shipname FROM ais_data.static_ships;
```

The table `static_ships` contains duplicated information about ships, transmitted at different timestamps, because nominative messages (ITU 5, ITU 19, and ITU 24) are automatically repeated at regular time intervals. The duplicated rows

in this table can be grouped together using a SQL view, as shown in the following example.

SQL *Q4.2.3*

```
CREATE OR REPLACE VIEW ais_data.ships AS
SELECT
   sourcemmsi, -- MMSI identifier of the ship, attributed by country of flag
   imo, -- IMO number of the ship, linked to the vessel structure
   callsign, -- Callsign of the ship
   shipname -- Ship name
FROM ais_data.static_ships
GROUP BY sourcemmsi, imo, callsign, shipname;
```

Using the view `ais_data.ships`, let assess some quality aspects of static AIS information, as proposed in the following exercise.

Do it yourself ! *A4.2.1*

How many ships have null, empty or space filled `shipname` ? [Answer: 14]
How many MMSI numbers are used more than once? [Answer: 327]

The view `ais_data.ships` and the queries proposed above show that the static ship information is not always accurate, because both fields `shipname` and `sourcemmsi` have duplicates and null or empty values. Indeed, this information can be manually modified. Being unreliable and inconsistent, none of these fields can used as a primary key in our tables.

In order to analyse the movement of ships, it is necessary to load their positions in the database. In the maritime dataset, vessel positions are stored in the file `nari_dynamic.csv`.

Do it yourself ! *A4.2.2*

Within the schema `ais_data`, create a new table named `dynamic_ships`. Fill it with the data of the file `nari_dynamic.csv`.

The following query counts the number of the ships positions contained in the table `dynamic_ships` ([Answer: 19,035,630]).

SQL *Q4.2.4*

```
SELECT COUNT(*) FROM ais_data.dynamic_ships;
```

4.3 The temporal dimension of data

PostgreSQL defines types, functions and operators to represent, handle and query dates and time based on Allen's interval algebra [1]. They enable, for instance, to test temporal relations and manipulate temporal events and intervals (e.g. the function `overlaps` checks if two temporal intervals overlap; "-" and "+" enable to add or subtract days or hours from a `time` or a `timestamp` value).

In the table `dynamic_ships`, the time of each vessel position is represented as an `integer` (in attribute `ts`), which expresses the corresponding epoch UNIX timestamp, i.e. the time, in seconds, elapsed since 1970-01-01 00:00:00 UTC (Coordinated Universal Time). In the following example, this numerical value is converted into `timestamp`, in order to facilitate its interpretation and to express time-oriented queries. In the following example, a new column is created to store the new representation.

SQL *Q4.3.1*

```
ALTER TABLE ais_data.dynamic_ships ADD COLUMN t timestamp without time zone;

UPDATE ais_data.dynamic_ships SET t=to_timestamp(ts);
```

This `timestamp` column facilitates visualising the temporal component of the table `dynamic_ships` and expressing temporal queries. The query in the example below asks for the temporal range of the table (i.e. minimum and maximum timestamp) ([Answer: "2015-10-01 00:00:01" - "2016-03-31 23:59:59"]).

SQL *Q4.3.2*

```
SELECT min(t), max(t) FROM ais_data.dynamic_ships;
```

4.4 Make it faster !

The queries of the previous section may take a while to complete. This is due to the size of the table `dynamic_ships` and because the table needs to be fully scanned for reading and sorting every timestamp in order to execute the queries. Querying very large tables can take a long time to execute. In order to optimise the queries, especially if they are executed frequently, the *PostgreSQL* commands `EXPLAIN` and `ANALYZE` can be used to understand how the DBMS plans to execute them. For example, the following command asks the *PostgreSQL* planner what is the execution plan for the last query in the previous section.

PostgreSQL *Q4.4.1*

```
EXPLAIN ANALYZE SELECT min(t), max(t) FROM ais_data.dynamic_ships;
```

In the answer, the *PostgreSQL* planner indicates that it would perform a sequential scan (`Seq Scan`) of the table. The sequential scan can be avoided if the column `t`, which appears in the `SELECT` clause of the query, is previously sorted and indexed. Indexes can require time to build up, and use storage space on the *PostgreSQL* server, but they can drastically decrease the time required to execute a query.

In the SQL example below, a Binary Tree (`BTREE`) index is created on column `t`. This index is particularly efficient for reading attributes whose values have a linear ordering like temporal, numerical or string attributes.

SQL *Q4.4.2*

```
CREATE INDEX idx_dynamic_ships_t -- name of the index
ON ais_data.dynamic_ships -- name of the table to index
USING btree (t) ; -- indexing technique (columns to index)
```

After creating the index, a new investigation of the query plan (as for the example above) shows that the `Index Scan` of the query would take a few milliseconds.

Remark !

In order to optimise the queries on very large tables, do not forget to define indexes on the columns that are often queried.

Do it yourself ! *A4.4.1*

Find how many different ships have broadcast their positions on AIS on January 1st 2016 and display their ship names. Which tables are required to answer this question? Do you require indexes on these tables?

[Answer: 79 rows, 78 different ship names]

4.5 Make it geographic !

The geo-spatial component is an important feature of the maritime dataset. Ships move in a specific spatial context made of coastlines, straights, ports, mooring areas, restricted areas and bathymetry. This information should be considered when analysing the maritime situation. In order to support this variety of geographic objects, *PostgreSQL* can be extended with *PostGIS* that adds geographic types, functions and indexes to the DBMS. In order to activate the *PostGIS* extension on a *PostgreSQL* database, it is sufficient to execute the command below.

PostgreSQL *Q4.5.1*

```
CREATE EXTENSION postgis;
```

Geographic objects

A way to represent spatial objects, such as moving vessels, is to use simple geometric features such as points, lines and polygons along with extra alpha-numerical attributes, e.g. the name of the spatial features. This way of modelling spatial objects is called *vector* representation. The coordinates of spatial objects are expressed in a given Coordinate Reference System (CRS). Thanks to *PostGIS*, *PostgreSQL* can handle the following geometric data types:

- `POINT`, defined using spatial coordinates in a CRS;
- `LINESTRING`, ordered set of `POINT`s;

- `POLYGON`, defined based on a collection of rings, at least one outer and possibly inner rings (representing, e.g. enclaves in countries, reservoirs), given as `LINESTRING`s;
- `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, i.e. sets of geometric objects of the same base type;
- `GEOMETRYCOLLECTION`, a set of geometric objects of various nature.

In table `dynamic_ships`, vessel positions are given by their coordinates, i.e. longitude and latitude, specified in two separated columns, `lon` and `lat`, respectively. The *PostGIS* function `ST_MakePoint` creates a `POINT` from two coordinates, given their CRS. In the example below, the World Geodesic System 1984 (WGS84) is used. It is identified in *PostGIS* through its Spatial Reference Identifier (SRID) as defined by the European Petroleum Survey Group (EPSG:4326[20]). The World Geodetic System 1984 is extensively used to define worldwide locations by longitude and latitude. This CRS is used by the GPS satellite navigation system integrated into AIS transceivers. The *PostGIS* function `ST_SetSRID` is used to specify the CRS' SRID of the newly created column. As the table `dynamic_ships` has 19 million positions, the update process may take a while (about 10 minutes).

SQL *Q4.5.2*

```
ALTER TABLE ais_data.dynamic_ships ADD COLUMN geom geometry(Point,4326);

UPDATE ais_data.dynamic_ships
SET geom=ST_SetSRID(ST_MakePoint(lon,lat),4326);
```

The database must be aware of the data's CRS before executing the commands in the example above. Data's CRS must be listed in the system table `spatial_ref_sys`, which is included in the *PostgreSQL* database public schema. The public schema is automatically created in every *PostgreSQL* database and contains system tables and functions. It is also used to store all the user-defined tables that are created without referring to any specific schema.

Remark !

Geographic positions on Earth can also be projected on a flat plane, introducing some distortions. Each map projection preserves important properties (direction, angle, shape, area, distance, *etc.*) while distorting others. Once projected, distances between objects can be computed using the reference metric of the map projection (e.g. meters). Each CRS is usually customised for a specific area. The maritime dataset covers Europe and provides data expressed using CRS WGS84, and ETRS89 / LAEA Europe (EPSG:3035[a]). The CRS ETRS89/LAEA Europe typically suits statistical mapping at all scales and covers the European Union (EU) countries onshore and offshore.

[a] `https://epsg.io/3035`

PostGIS can project geometric objects from one CRS to another. In the example below, the function `ST_Transform` is used to project vessel positions expressed in WGS84 into the projection ETRS89/LAEA Europe CRS (EPSG:3035).

[20] `https://epsg.io/4326`

SQL *Q4.5.3*

```
ALTER TABLE ais_data.dynamic_ships ADD COLUMN geom3035 geometry(Point,3035);

UPDATE ais_data.dynamic_ships
SET geom3035=ST_Transform(geom,3035);
```

4.6 Integrating contextual data

The maritime dataset contains also complementary data like ports' positions, restricted and protected areas, coastlines, weather and ocean conditions. These data have a different nature than AIS data, which are continuously streamed, and provide information useful to contextualise and understand the vessel movements. Conveniently, a dedicated database schema, with name `context_data`, can be created for these data.

Action required

Create a new schema `context_data` in the database `maritime_informatics`.

Loading spatial *Shapefile* data

All contextual (geographic) data in the dataset are in the same format, i.e. *Shapefile*, which is a well-known format for vector (i.e. geometric) data. *Shapefile* data are stored in multiple files, which have the same name and different extensions and represent different geometric aspects, as follows:

- SHP: is the shape of the geometric objects;
- DBF: contains alpha-numerical attributes;
- PRJ: is the coordinate reference system of the geometric objects;
- SHX: is a shape geometric index.

Do it yourself ! *A4.6.1*

Download the file "[C2] European Coastline.zip" from the dataset. Unzip the folder in your working directory and look at the different files. Have a look at the PRJ file. What is the coordinate reference system of the "Europe Coastline (Polygon)" shapefile (SHP)?

[Answer: GRS 1980]

Shapefile data can be imported in a spatially enabled *PostgreSQL* database using the *PostGIS* tool `shp2pgsql`. The example below uses `shp2pgsql` to load the *Shapefile* data stored in `Europe Coastline (Polygone).shp`. Data are loaded into the database `maritime_informatics`. A coordinate system (ETRS89 / ETRS-LAEA, 3035) must be specified to correctly import them. Figure 3 shows the interface of the `shp2pgsql` tool.

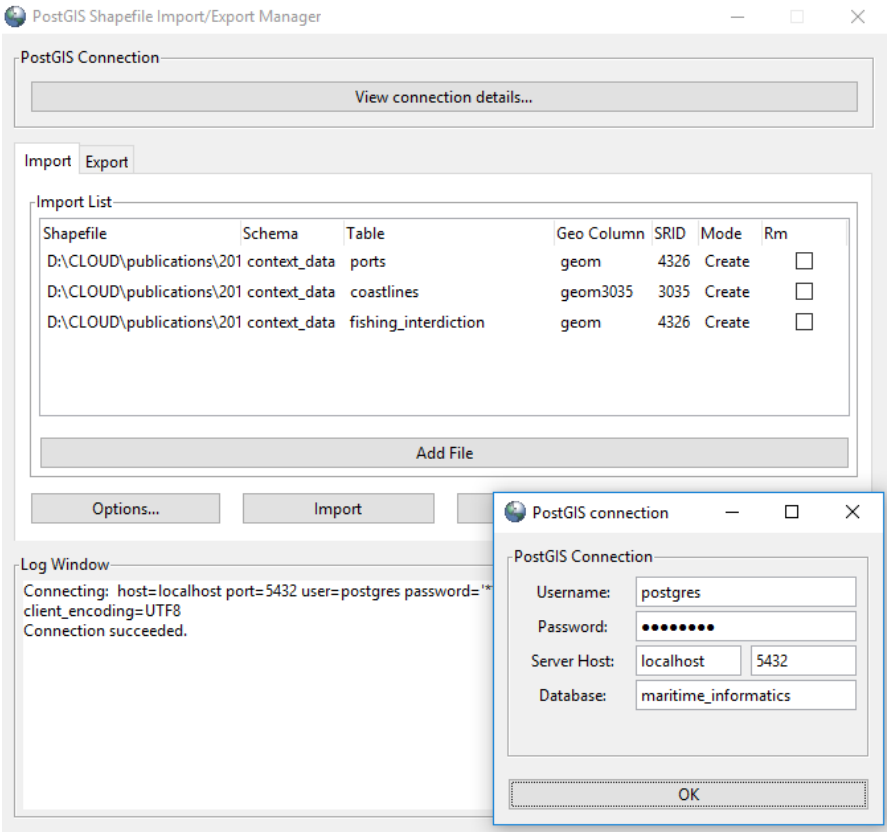

Fig. 3 The `shp2pgsql` interface is used to load *Shapefile* data into a spatially enabled *PostgreSQL* database.

Action required

In order to load the European coastline in the `maritime_informatics` database, execute the following steps:

1. start the `shp2pgsql` interface (`shp2pgsql-gui`);
2. load the file `Europe Coastline (Polygone).shp`;
3. rename the target table into `coastlines`;
4. select the schema `context_data`;
5. set the coordinate system to ETRS89 / ETRS-LAEA (3035);
6. once the import parameters are set up, import the data.

Upon a successful upload, *pgAdmin* can be used to preview the first 100 rows of the newly populated table.

Action required

Using *pgAdmin*, to preview the first 100 rows of table `coastlines`:

1. right click on the table `coastlines`;
2. select *View/Edit data*, then *First 100 rows*.

Remark !

In *pgAdmin*, the "eye" button on a geometric column header enables to see the geometric shape of the first 100 rows in a spatial table.

Action required

In *pgAdmin*, click on the "eye" button on the column `geom3035` of the table `coastlines` to visualise their geometric shape. Now, start *QGIS*, connect to *PostgreSQL* database using the *Data Source Manager* then, add the `coastlines` geographic layer in a new map.

Readers having trouble to connect *QGIS* to *PostgreSQL* can refer to QGIS user manual.[21]

Do it yourself ! ***A4.6.2***

Using *pgAdmin*, find the type of the column `geom3035` in the table `coastlines`. What is the primary key of the table?

[Answer: `MULTIPOLYGON` - column `gid`]

Action required

From the dataset, select the following *Shapefiles* and load them in the database `maritime_informatics` as explained above:

- [C1] Ports of Brittany (EPSG:4326);
- [C4] Fishing Areas (European commission) (EPSG:4326);
- [C5] Fishing Constraints (EPSG:4326).

Once loaded, as for the examples in Section 4.5, add to all tables a new geometric column named `geom3035` and project the geometric data into the CRS EPSG 3035.

Action required

In *QGIS*, add the ports, fishing areas and fishing constraints geographic layers in your map as illustrated in Figure 1. Layers can be ordered using the *QGIS* Layer Panel. Layers should be positioned wisely. To be visible, the Port layer should be displayed above the coastline layer.

4.7 Executing spatial queries

Spatial queries evaluate the relationships that hold among objects' geometries. *PostGIS* supports the 8 region connection calculus (RCC8) [40] and the Dimensionally Extended nine-Intersection Model (DE-9IM) [11], and complies with the OGC

[21] `https://docs.qgis.org/3.4/en/docs/user_manual/managing_data_source/opening_data.html#database-related-tools`

Simple Feature Access ([20]), which defines the supported routines to test spatial relationships between two geometric objects. *PostGIS* functions that output a value, either a distance (i.e. `ST_Distance`) or a Boolean value (all the others) are shown in Figure 4. *PostGIS* functions that output a geometry are shown in Figure 5. All functions can be applied between geometries of various types (i.e. `POINTS`, `LINESTRINGS` or `POLYGONS`), as far as the coordinates of these objects are defined in the same CRS. These spatial relationship functions can be combined with temporal operators, enabling to express complex spatio-temporal queries.

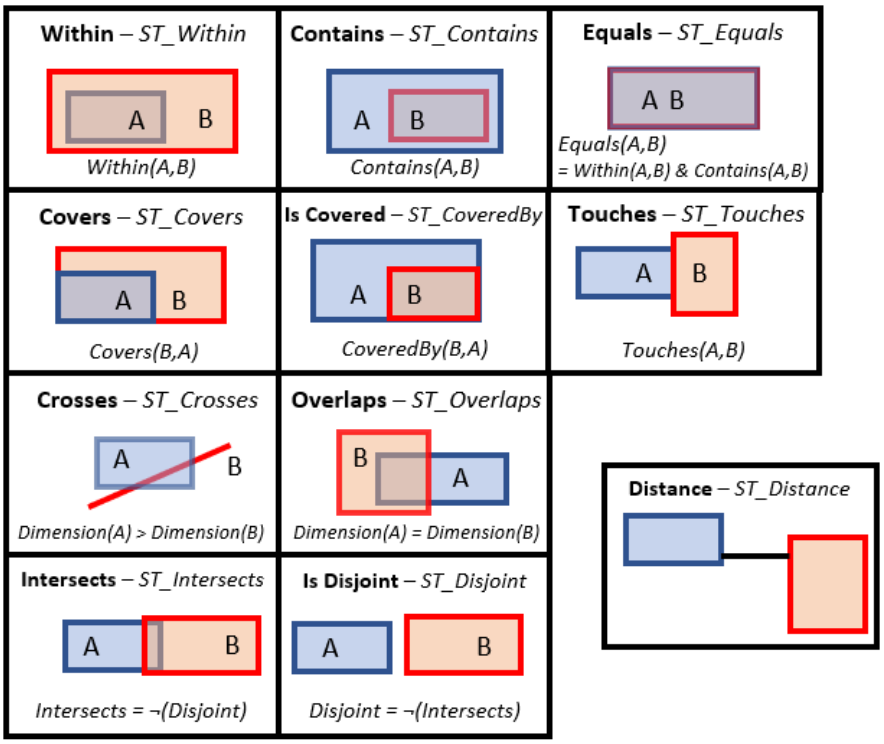

Fig. 4 *PostGIS* functions that output a numerical (distance) or a boolean value. On top of each box, the *PostGIS* function and the corresponding spatial relation are reported (**relation** - *PostGIS* function).

The functions `ST_Equals`, `ST_Disjoint`, `ST_Intersects`, `ST_Touches`, `ST_Crosses`, `ST_Within`, `ST_Contains`, and `ST_Overlaps` evaluate whether the corresponding spatial relationship holds between two input geometries, and return a boolean value. The function `ST_Distance` returns a real number representing the minimum distance between the two geometries. Other functions return the geometries resulting from the application of geometric operations (`ST_Buffer` and `ST_ConvexHull`) (cf. Figure 5) or set-based operations (e.g. `ST_Intersection`, `ST_Difference`, `ST_Union`, `ST_SymDifference`) (cf. Figure 6) . An exten-

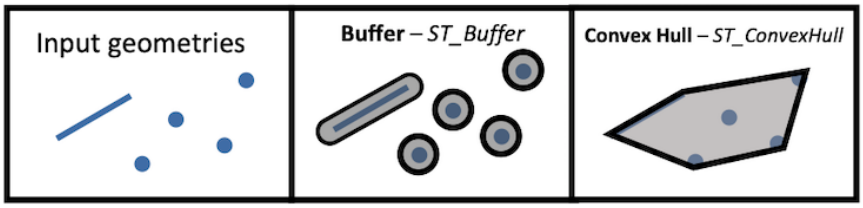

Fig. 5 Examples of *PostGIS* functions that output a spatial geometry (buffer and convex hull). Output geometries are shown in grey. On top of each box, the *PostGIS* function and the corresponding spatial function are reported.

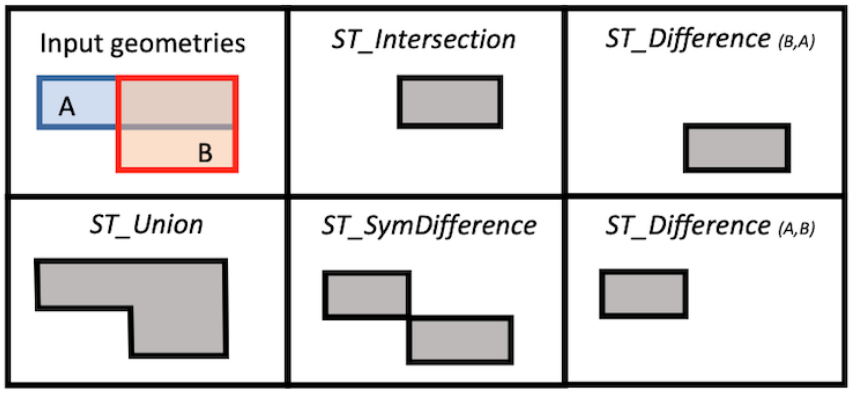

Fig. 6 Example of *PostGIS* functions that apply set-based operations and output a geometry. Output geometries are shown in grey. On top of each box, the corresponding *PostGIS* function is reported.

sive presentation of all these geometry processing functions is available on *PostGIS* website.[22]

The query in the following SQL example illustrates the use of such spatial functions. It searches for ships (name and unique identifier, i.e. the ship MMSI) that stopped within 500m from a port in Brittany on January 1st, 2016. For each ship, the name of the port and the time spent by the vessel in the port is also shown.

Remark !

The distance between ship positions and port locations is calculated using the *PostGIS* function `ST_dWithin`. The distance threshold (500m) can be visualised by creating a buffer around each port location using the function `ST_Buffer`.

[22] `https://postgis.net/docs/reference.html#Geometry_Processing`

SQL Q4.7.1

```
SELECT port_name, mmsi, shipname, min_t, max_t, (max_t-min_t) as dur
FROM (
    SELECT libelle_po as port_name, mmsi, min(t) as min_t, max(t) as max_t
    FROM context_data.ports as q1 -- ports location
    INNER JOIN ais_data.dynamic_ships as q2 -- ship AIS positions
    ON (
        speed=0  -- not moving
        AND t>='2016-01-01 00:00:00'
        AND t<'2016-01-02 00:00:00'  -- during Jan 1, 2016
        AND ST_dWithin(q1.geom3035,q2.geom3035,500) -- ships by 500m of port
    )
    GROUP BY libelle_po, mmsi ) as q3
LEFT JOIN ais_data.static_ships as q4 -- ship names
ON (q3.mmsi=q4.sourcemmsi);
```

Spatial indexes

Similar to *classical* SQL queries, spatial queries can be optimised using indexes applied to spatial columns. *PostGIS* supports the *GiST* index, which indexes the *bounding boxes*, or minimum rectangles, enclosing the spatial extent of the geometries in a geometric column. For instance, the bounding boxes of the geometries in Figure 7.a are depicted in Figure 7.b as gray, dotted rectangles, and highlighted by coloured rectangles in Figure 7.c. The example below shows how to create a *GiST* index on the geometry column `geom3035` of table `dynamic_ships`.

SQL Q4.7.2

```
CREATE INDEX idx_dynamic_ships_geom3035
ON ais_data.dynamic_ships -- table name
USING GiST (geom3035); -- GiST Index on geometric column
```

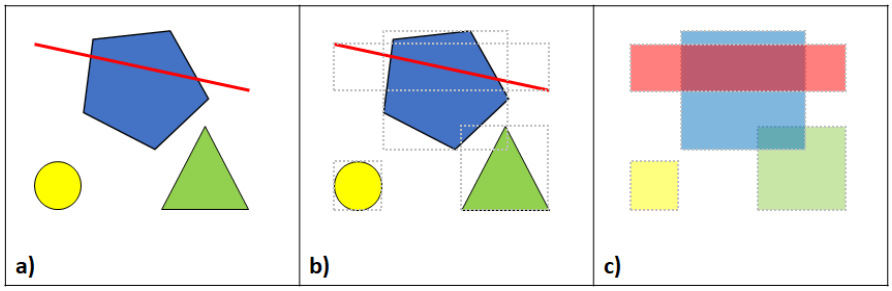

Fig. 7 Use of the GiST index for computing spatial queries. When geometries (a) are indexed (c), spatial queries use the geometries' bounding boxes to approximate the query.

Bounding boxes allow for a compact, while simplified, representation of geometries, that can be used for efficiently comparing the geometries in spatial queries. When a SQL query refers to an indexed geometry, a spatial relationship is firstly evaluated using the spatial extent of the geometry (Figure 7.c). Afterwards, only for

the subset of the potentially matching geometries, the exact spatial relationship is processed and refined. In the example of Figure 7, a query on (a) searching for intersecting geometric objects would exclude the yellow circle, because its bounding box does not intersect with any other geometric object's bounding box. The precise computation of the intersection is performed only between the blue pentagon and both the red line and green triangle (as they are the only boxes to overlap). With the index, the computation of the complex polygon intersection requires only two comparisons (first between the blue pentagon and the red line, then between the blue pentagon and the green triangle). Checking every geometric object with each other would have required $\binom{4}{2} = 6$ comparisons.

Do it yourself ! ***A4.7.1***

Execute again the SQL query of the previous example (looking for the ships that stopped at less than 500m from a port in Brittany on January 1^{st}, 2016). After creating a spatial index on column `geom3035`, you should get the answer faster than without the spatial index. Then, search for the vessels (names and identifiers) that fished more than 15 minutes on January the 22^{nd} 2016 within any fishing area. Usually, a fishing vessel fishes at a speed between 2.5 and 3.5 knots.

[Answer: 7 vessels].

4.8 Extending PostgreSQL functions

As described in the previous section, *PostGIS* defines functions to manipulate and query geographic data. These functions extend the set of data management functions provided by *PostgreSQL*. Novel functions may be created using procedural programming languages (PL) like *PL/pgSQL*, *PL/Python*, *PL/Tcl*, *PL/Perl*, *PL/Java*, *PL/R*, *PL/sh*.[23]

Remark !

You can view the *PostgreSQL* and *PostGIS* functions in the folder "Functions" which is included in every database schema.

In the following example, a *PL/pgSQL* function is defined to search for the name and the identifier, i.e. the IMO number, of a vessel, given its MMSI. The function returns the result in formatted text.

[23] `https://www.postgresql.org/docs/11/xplang.html`

SQL *Q4.8.1*

```
CREATE OR REPLACE FUNCTION get_vessel_info( -- name of the fuction
  mmsi integer -- list of input argument with types
)
RETURNS text AS $$ -- type of the returned value
DECLARE
    vessel_imo integer; -- integer local variable
    vessel_name text; -- text local variable
BEGIN
    SELECT shipname, imo INTO vessel_name, vessel_imo
    FROM ais_data.static_ships
    WHERE sourcemmsi=mmsi LIMIT 1;
    RETURN '[' || vessel_imo || '] ' || vessel_name; -- return value
END;
$$ LANGUAGE plpgsql; -- programming language used
```

Once defined, the function can be called in a query, as shown below.

SQL *Q4.8.2*

```
SELECT get_vessel_info(227705102);
```

[Answer: "[262144] BINDY"]

5 Understanding vessel movements with trajectory-based queries

A trajectory can be defined as "a record of the evolution of the position (perceived as a point) of an object moving in space during a given time period" [52]. Position-based queries (i.e. relying on geometric data of type `POINT`) are easy to formalise and can provide meaningful information and statistics about these movement data. However, they have several limitations. First, there is a computational limit, as such spatial queries are very expensive. Second, the information they provide is sometimes limited by the update rate or the coverage of the sensor that measures the object's position.[24] In the maritime domain, for instance, it is difficult to identify with certainty a vessel that has crossed a narrow passage, in order to check whether it has entered a restricted area or to calculate exactly the minimum distance to the coast.

The notion of *trajectory*, for instance as discussed in [36], whilst sometime complex to implement, has been introduced to address these limitations. It underlies the use of filtering and clustering techniques that make it possible to clearly define the starting, intermediate and ending points of a trajectory.

In this section we will create vessel trajectories by connecting the points of the same ship between them, in the form of polylines. This is a simple implementation of a *stop-move* model of trajectories [52]. Ship trajectories are segmented using time intervals. During these time intervals, the ship positions can stay still (stop) or change (move).

[24] In the case of AIS, vessel positions are reported only in the areas covered by AIS receivers, and at sparse time intervals. The AIS data in the open dataset are collected from a terrestrial receiver.

In *PostGIS*, the *move* part of the trajectory can be represented as a `LINESTRING`, which connects sequences of positions. In order to derive stationary (*stop*) areas, multiple AIS positions of anchored ships may be spatially grouped together, which can be spatially represented by cluster centroids. The overall ship movements can be modeled with a graph, whose nodes depict the stop locations (e.g. ports, mooring areas), and edges link stops. Ship trajectories can be grouped along edges and aggregated using statistics. This *node-edges* model can itself be manipulated, queried, analysed, for instance to analyse ships' life cycle as shown in [21].

In order to store the trajectories and the results of the data analysis queries that will be presented in the following sections, the creation of a new schema named "data_analysis" is required.

Action required

Define a schema `data_analysis` to store all the results of the data analysis queries.

5.1 Characterising port areas through spatial partitioning

Spatial partitioning enables to understand the essential characteristics of movement data. With this aim, Andrienko & Andrienko [3] partition movement data according to a *Voronoi tessellation* [4]. A Voronoi diagram is a partition of a plane into regions (Voronoi cells) around a set of seed points. The voronoi cell around a seed point encompass all points of the plane closer to that seed point.

In this section, a Voronoi tessellation will be used to get an approximation of the area of competence of each port. In most official databases, ports are represented by geometrical points. This is also the case of the data in table `ports` in schema `context_data`. This representation is not effective, in extracting, for instance, from the database the vessels that are stopped in a port at a certain instant in time. In order to answer this query, ports should be represented by geometrical regions, on which e.g. the *PostGIS* function `ST_Within` could be applied. In order to derive ports' regions, a space partitioning must be created. Note that, since ports are not equally distributed along the coast, a uniform partition of the space would not be accurate to answer the query above.

By constructing a Voronoi tessellation of the space based on port locations, vessel positions can be dynamically associated to ports. The Voronoi Polygons around a set of points can be computed using *PostGIS* `ST_VoronoiPolygons` function. For instance, given the tessellation in Figure 8, stopped vessels (i.e. with null or negligible speed over ground) can be associated to the closest ports using the spatial operators described above. Afterwards, the distance between the stop location and the port can be computed.

In the example below, a new table (`ports_voronoi`) is created to contain the Voronoi tessellation based on port locations. A *GiST* index is created to optimise the access to the newly created geometry.

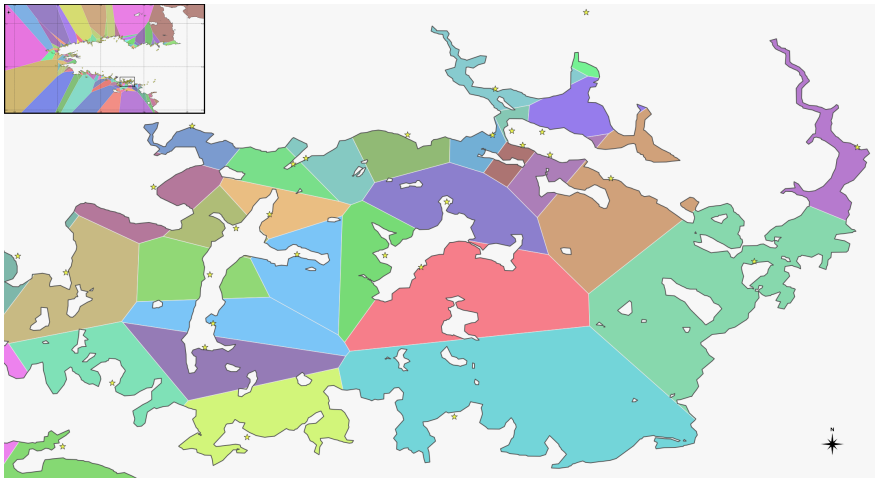

Fig. 8 Voronoi tessellation of the geographical space, partitioning sea and land to identify the nearest port of any maritime location in Gulf of Morbihan. The Voronoi cells have random colours. Land is depicted in white, with the black line representing the coastline. Yellow stars represent ports' positions.

SQL *Q5.1.1*

```
CREATE TABLE data_analysis.ports_voronoi AS
    SELECT por_id as port_id, libelle_po as port_name, geom3035,
     voronoi_zone3035
    FROM context_data.ports
    LEFT JOIN (
        SELECT (ST_Dump(ST_VoronoiPolygons(ST_Collect(geom3035)))).geom
            as voronoi_zone3035
        FROM context_data.ports) as vp
    ON (ST_Within(ports.geom3035, vp.voronoi_zone3035));

CREATE INDEX idx_ports_voronoi_zone ON data_analysis.ports_voronoi
USING gist (voronoi_zone3035);
```

The following SQL query creates a new table (`non_moving_positions`) dedicated to the storage of stopped vessel positions. Thanks to the Voronoi tessellation in table `ports_voronoi`, each stop position can be matched to a unique port. Once matched, the distance between a vessel stop and the associated port can be computed using the built-in function `ST_Distance`.

SQL *Q5.1.2*

```
CREATE TABLE data_analysis.non_moving_positions AS
    SELECT id, mmsi, t, q1.geom3035, port_id, port_name,
    ST_Distance(q1.geom3035,ports_voronoi.geom3035) as port_dist
    FROM (
        SELECT *
        FROM ais_data.dynamic_ships
        WHERE speed=0 -- non moving ship positions only
        ) as q1
    LEFT JOIN data_analysis.ports_voronoi
    ON ST_Within(q1.geom3035,ports_voronoi.voronoi_zone3035); -- ships in
     voronoi area
```

With the execution of the following queries, additional indexes are created to optimise the access to this new table `non_moving_positions` (on position geometry, position identifiers and associated timestamp).

SQL *Q5.1.3*

```
CREATE INDEX idx_non_moving_positions_geom3035
ON data_analysis.non_moving_positions
USING gist (geom3035);

CREATE INDEX idx_non_moving_positions_port_id
ON data_analysis.non_moving_positions
USING btree(port_id);

CREATE INDEX idx_non_moving_positions_t
ON data_analysis.non_moving_positions
USING btree (t);
```

Do it yourself ! *A5.1.1*

Using the results of table non_moving_positions, compute the average distance of vessel stops in the Voronoi area of the Brest port.

[Answer: 1434.75m]

5.2 Detecting and clustering ship stops

As it will be illustrated in the chapter of Andrienko et al. [2], clustering techniques can be applied to group vessels stops outside ports and for detecting the different docking areas within a port. In the example of the previous section, many vessel stops are located outside the ports, in areas which are likely mooring areas.

5.2.1 Detecting ship stops

The analysis of vessel movement begins with the identification of stops. A stop can be defined as a temporally ordered sequence of vessel positions with speed (i.e. speed over ground, SOG) equal to zero or below a very low threshold. The queries presented in the next examples create, for each vessel, the sequences of stops extracted from the history of the vessel's positions.

To build this sequence, first, an auxiliary table of successive position pairs, namely `segments` is created. For each ship, this table enables us to connect with a line segment every consecutive pair of AIS positions. The table is ordered according to the MMSI number and the position timestamp (`ORDER BY mmsi, t`), and the *PostgreSQL* function `LEAD` will be used to access the data in the next row of an ordered table.

From this table, for each vessel, stop events can be extracted and filtered according to the vessel speed that is associated to the preceding and the following positions of each segment. This table is also useful to detect transmissions gaps between vessel positions (*cf.* the chapter of Patroumpas et al. [37]) and GPS malfunctions (e.g. detecting unfeasible speeds. A new index combining the vessels' MMSI and the timestamp of the vessels' positions is created to optimise the query. The SQL commands for creating the index and the table are given in the example below.

SQL *Q5.2.1*

```
CREATE INDEX idx_dynamic_ships_mmsi_t ON ais_data.dynamic_ships
USING btree (mmsi,t);

CREATE TABLE data_analysis.segments AS
    SELECT mmsi, -- ship identifier
        t1, t2, -- starting and ending timestamps
        speed1, speed2, -- starting and ending speeds
        p1, p2, -- starting and ending points
        st_makeline(p1,p2) as segment, -- line segment connecting points
        st_distance(p1,p2) as distance, --  distance between points
        extract(epoch FROM (t2-t1)) as duration_s, -- timestamps in seconds
        (st_distance(p1,p2)/extract(epoch FROM (t2-t1)))
         as speed_m_s -- speed in m/s
    FROM (
        SELECT mmsi, -- ship identifier
          LEAD(mmsi) OVER (ORDER BY mmsi, t) as mmsi2, -- next MMSI
          t as t1, -- starting time
          LEAD(t) OVER (ORDER BY mmsi, t) as t2, -- ending time
          speed as speed1, -- initial speed
          LEAD(speed) OVER (ORDER BY mmsi, t) as speed2, -- final speed
          geom3035 as p1, -- initial point
          LEAD(geom3035) OVER (ORDER BY mmsi, t) as p2 -- final point
        FROM ais_data.dynamic_ships ) as q1
    WHERE mmsi=mmsi2; -- filter out different mmsi

CREATE INDEX idx_segments_speed ON data_analysis.segments
USING btree (speed1,speed2);
```

Once the table `segments` is created, stop events can be detected. Aligned with the chapter of Andrienko et al. [2], each stop starts immediately after a move, and ends as soon as the vessel starts moving again. Starting from the data in table `segments`, segments whose initial speed is above the speed threshold, and whose final speed is below the speed threshold, represent the beginning of stops (i.e. the vessel decreases its speed and goes steady). The end of a stop event is detected conversely.

The following example creates the auxiliary tables `stop_begin` and `stop_end`, which contain the potential starting and ending positions of vessel stops, selected as just described. In this example, a speed threshold of 0.1 knot is used. To optimise the future access to these tables, indexes are created.

SQL *Q5.2.2*

```
CREATE TABLE data_analysis.stop_begin AS -- first stop position
    SELECT mmsi, t2 as t_begin -- stop starts at first steady position
    FROM data_analysis.segments
    WHERE speed1>0.1 AND speed2<=0.1; -- speed threshold is 0.1 kn

CREATE INDEX idx_stop_begin_mmsi_t ON data_analysis.stop_begin
USING btree (mmsi,t_begin);

CREATE TABLE data_analysis.stop_end AS -- last stop position
    SELECT mmsi, t1 as t_end -- stop ends at last steady position
    FROM data_analysis.segments
    WHERE speed1<=0.1 AND speed2>0.1; --  speed threshold is 0.1 kn

CREATE INDEX idx_stop_end_mmsi_t ON data_analysis.stop_end
USING btree (mmsi,t_end);
```

Afterwards, the table `stops` is created by coupling, for each ship, `stop_begin` and `stop_end`, as illustrated in the following example (note that stops are ordered by time, that is, `ORDER BY t_end`).

SQL *Q5.2.3*

```
CREATE TABLE data_analysis.stops AS
    SELECT
        mmsi, -- ship identifier
        t_begin, -- start of the stop event
        t_end, -- end of the stop event
        extract(epoch FROM (t_end-t_begin)) as duration_s -- stop in seconds
    FROM data_analysis.stop_begin INNER JOIN LATERAL (
        -- keep only stops that have an end
        SELECT t_end
        FROM data_analysis.stop_end
        WHERE stop_begin.mmsi=stop_end.mmsi AND
            t_begin<=t_end -- stop follows the beginning
        ORDER BY t_end LIMIT 1 --  select only the first stop end
    ) AS q2 ON (true);
```

In the example above, `LATERAL` is a reserved *PostgreSQL* keyword that enables cross-references between the main query and the subquery (at the right of `LATERAL`. In this case, it is used to compare the time of `stop_begin.t_begin` with `stop_end.t_end`.

Not all the detected stops are meaningful. For instance, stops that last for a short time, or are spatially too spread, could be filtered out. Similarly, stops with few vessel positions can be discarded. In the example below, the centroid of each vessel' stop cluster and the number of associated positions are computed.

SQL *Q5.2.4*

```
ALTER TABLE data_analysis.stops ADD COLUMN centr3035 geometry(Point,3035);
ALTER TABLE data_analysis.stops ADD COLUMN nb_pos integer;

-- compute the centroid and number of positions
UPDATE data_analysis.stops SET (centr3035, nb_pos) = (
    SELECT st_centroid(st_collect(geom3035)), -- centroid of a multipoint
      count(*) as nb -- number of points
    FROM ais_data.dynamic_ships
    WHERE mmsi=stops.mmsi AND
        t>=stops.t_begin AND t<=stops.t_end);  -- timestamp range
```

In the following example, indicators on the dispersion of the vessel's positions around the centroid of the cluster they belong to are computed. Once aggregated, this information offers some intuition about the nature of the stop. Additional columns are created in the table `stops` to store the computed indicators (average and maximum distances from the centroid).

SQL *Q5.2.5*

```
ALTER TABLE data_analysis.stops ADD COLUMN avg_dist_centroid numeric;
ALTER TABLE data_analysis.stops ADD COLUMN max_dist_centroid numeric;

-- compute the distance of the position cluster to the centroid
UPDATE data_analysis.stops
SET (avg_dist_centroid, max_dist_centroid) = (
    SELECT avg(d), max(d)
    FROM (
        SELECT st_distance(centr3035,geom3035) as d -- distance to centroid
        FROM ais_data.dynamic_ships
        WHERE mmsi=stops.mmsi AND
            t>=stops.t_begin AND t<=stops.t_end -- timestamp range
        ) as q1);
```

In order to illustrate to use of the structures built above, the next query calculates how many of the vessel' stops lasted more than 5 minutes, involved more than 5 consecutive positions and occurred in average within 10 m of the centroid of the vessel' stop cluster.

SQL *Q5.2.6*

```
SELECT count(*)
FROM data_analysis.stops
WHERE duration_s >=(5*60) -- 5 minutes long
    AND nb_pos>5 -- minimum 5 consecutive positions
    AND avg_dist_centroid <=10; -- within 10m from centroid
```

[Answer: 22,987]

5.2.2 Clustering ship stops

The auxiliary structures created in the previous section calculated, for each vessel, its stop clusters, i.e. the areas where the vessel was steady. In this section a density-based algorithm will be used to detect stationary areas.

Density-based clustering algorithms [34, 56] can be used to extract from data stationary areas, turning points in trajectories, activities areas, and so on. *PostGIS* implements the well-known *Density-Based Clustering of Applications with Noise* algorithm (DBSCAN) [12]. This algorithm clusters geometric objects together provided there are more than a minimum number (n) of neighbouring objects located within a threshold distance (d). These parameters should be tailored to the area under consideration.

In the following example, DBSCAN is used to cluster stop centroids in order to detect frequent stop areas. In the example, n defines the minimum number of stops to form a cluster; d defines the minimum distance between stops belonging to two different clusters. The smaller the d and n parameters, the bigger the number of

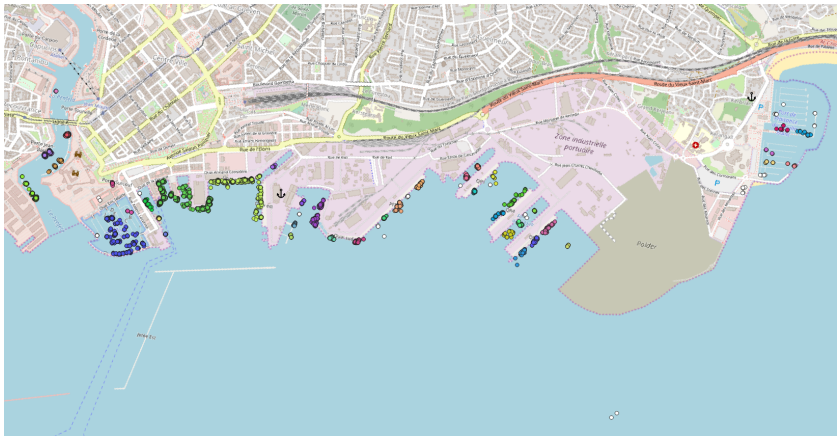

Fig. 9 DBSCAN clustering of vessel stops (random colours, one per cluster) in Brest.

resulting clusters. If d is large, then multiple mooring areas on a same dock will be clustered together. A similar approach is also presented in the chapter of Andrienko et al. [2] to detect stopping areas.

In the example below, groups of stops lasting more than one minute and having at least 5 different occurrences within 50 m distance range are clustered and stored in table `cluster_stops`. DBSCAN generated 354 different clusters that can be visualised using *QGIS*. These clusters are precise enough to detect different mooring locations within big ports like the one of Brest as illustrated in Figure 9.

SQL *Q5.2.7*

```
CREATE INDEX idx_stops_centroid ON data_analysis.stops
USING btree (centr3035);

CREATE TABLE data_analysis.cluster_stops AS
    SELECT *, ST_ClusterDBSCAN(centr3035, eps := 50, minpoints := 5) OVER ()
      AS cid
    FROM data_analysis.stops
    WHERE duration_s >=60; -- 1 minute long
```

In order to to define the location of frequently used docking and mooring areas, the *spatial hull* of these clusters can be computed.

The convex (respectively, concave) hull of a set of points represents the minimum convex (respectively, concave) geometry that encloses all the points within the set. The Minimum Bounding Circle represents the smallest circle that fully contains all the points of the set. All these polygons can be computed using various *PostGIS* functions.

The area of the spatial hull can also give hints about the clustering process quality. Clusters with high spatial dispersion will have a bigger polygon surface. This can be acceptable for mooring areas outside a port, but within a port smaller clusters are likely more adequate to match docking areas.

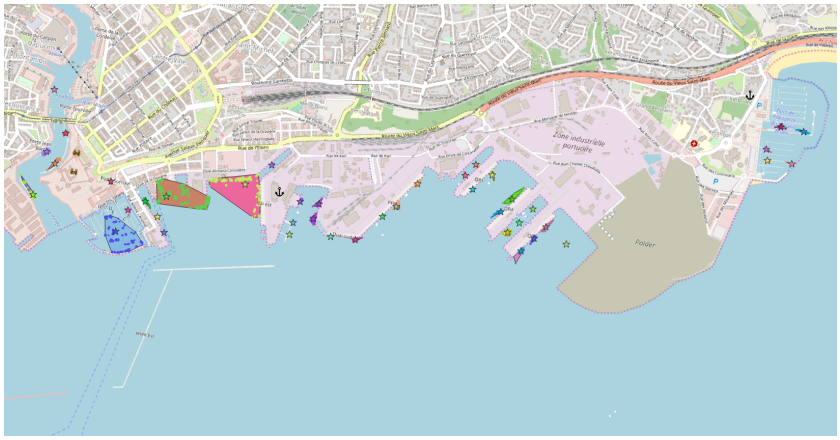

Fig. 10 Convex hulls of vessel stop clusters (random colours) in the area of the Brest port.

The following query shows how to compute different spatial hulls of vessel stops. For each cluster, statistics are also included. In Figure 10, the convex hulls of vessel stops in the port of Brest are visualised with *QGIS*.

SQL *Q5.2.8*

```
CREATE TABLE data_analysis.clusters_stops_hulls AS
    SELECT cid, -- cluster id
        ST_ConvexHull(st_collect(centr3035)) as convex_hull,
        ST_ConcaveHull(st_collect(centr3035),0.75) as concave_hull,
        ST_MinimumBoundingCircle(st_collect(centr3035)) as
                bounding_circle,
        ST_Centroid(st_collect(centr3035)) as centroid,
        count(*) as nb_stops, -- number of stops in this cluster (area)
        sum(nb_pos) as nb_pos, -- sum of stops in the cluster
        count(DISTINCT mmsi) as nb_ships, -- num of unique ships
        min(duration_s) as min_dur,
        avg(duration_s) as avg_dur,
        max(duration_s) as max_dur
    FROM data_analysis.cluster_stops
    WHERE cid IS NOT NULL -- exclude outliers
    GROUP BY cid; -- group all the stops centroids within the same cluster
```

5.3 Extracting trajectory tracks and the navigation graph

Maritime traffic can be modelled as a graph whose nodes correspond to the stationary areas (e.g. mooring or docking areas) and edges represent the vessel movements between these areas. Relying on such a graph, vessel movement can be analysed [21, 29, 10]. For example, it is possible to count and identify the incoming and outgoing destinations from each stationary area, to calculate the traffic density, and to identify the most frequently used tracks.

In the examples that follow, and similarly to the computation of stops, vessel tracks will be created considering the successive positions of the same ship between a *track start* and a *track stop*. By definition:

- the *start* of a vessel track is the end of the previous stop event; and
- the *end* of a track is the start of the next stop event.

In the following example, a new table `tracks` is created and populated with stop events selected from the table `cluster_stops`, which already contains all the filtered stops with associated cluster identifiers. These will be used as the nodes of the vessel traffic graph. An index is added to optimise the selection of temporally ordered points.

SQL *Q5.3.1*

```
CREATE INDEX idx_cluster_stops_mmsi_tbegin ON data_analysis.cluster_stops
USING btree (mmsi, t_begin);

CREATE TABLE data_analysis.tracks AS
    SELECT q1.mmsi, -- ship id
        q1.cid as start_cid, -- start cluster node id of the track
        q3.cid as end_cid, -- end cluster node id of the track
        q1.t_end as t_start, -- track start is end of the previous stop
        q3.t_begin as t_end, -- track end is begin of the next stop
        extract(epoch FROM (q3.t_begin-q1.t_end))
            as duration_s -- track duration in seconds
    FROM (
        SELECT mmsi, cid, t_end
        FROM data_analysis.cluster_stops
        WHERE cid IS NOT NULL ) as q1
        INNER JOIN LATERAL (
          -- track that are in between two clustered stops area
            SELECT q2.cid, q2.t_begin
            FROM data_analysis.cluster_stops as q2
            WHERE q2.cid IS NOT NULL -- stop must be in a clustered area
            AND q1.mmsi=q2.mmsi -- same ship
            AND q2.t_begin>q1.t_end -- search for the next stop event
            ORDER BY q2.t_begin LIMIT 1 ) as q3 ON (true);
```

As before, the reserved word `LATERAL` enables to cross reference the elements of the first sub-query with alias `q1` in the second sub-query `q2` (`q1.mmsi=q2.mmsi AND q2.t_begin>q1.t_end`).

In the next query, the geometry of each *track move* is represented as a segment connecting consecutive track stops. Ship positions between the `t_start` and `t_end` timestamps of the trajectory are selected, ordered with respect to time and connected to form a `LINESTRING` using the `ST_makeline` function of *PostGIS*. Figure 11 shows the computed trajectory tracks.

SQL *Q5.3.2*

```
ALTER TABLE data_analysis.tracks
ADD COLUMN track geometry(LineString,3035);

UPDATE data_analysis.tracks SET track = (
    SELECT st_makeline(geom3035)
    FROM (
        SELECT geom3035
        FROM ais_data.dynamic_ships
        WHERE mmsi=tracks.mmsi AND t>=tracks.t_start AND t<=tracks.t_end
        ORDER BY t) as q1); -- order points by time
```

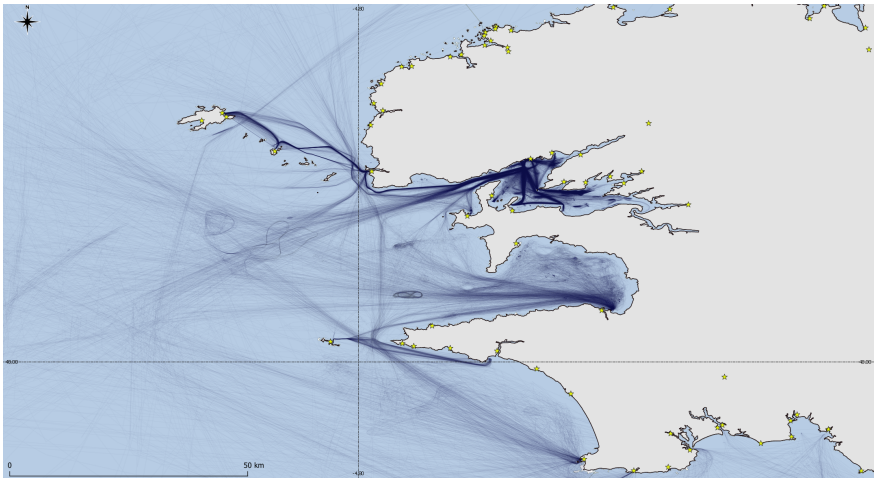

Fig. 11 Vessel trajectory tracks in the Brest area (yellow stars are ports of Brittany).

The SQL example that follows creates a new table (`graph_edges`) to represent the traffic graph of the Brest area. For each graph edge, statistics on the underlying traffic, like the number of associated vessel tracks, the number of ships navigating it, the average trajectory duration and length, are extracted. These indicators are stored in the table and can be visualised for analysis. The traffic along the edges of the graph is oriented.

SQL *Q5.3.3*

```
CREATE TABLE data_analysis.graph_edges AS
    SELECT
        start_cid, -- from node
        end_cid, -- to node
        nb_tracks, -- number of tracks
        q1.nb_ships, -- number of different ships
        q1.avg_duration, -- average transit time
        q1.avg_length, -- average transit length
        q1.min_length, -- minimum transit length
        st_makeline(c1.centroid,c2.centroid) as straight_edge -- edge
    FROM (
        SELECT
            start_cid, -- from node
            end_cid, -- to node
            count(*) as nb_traj, -- number of vessel trajectories
            count(distinct mmsi) as nb_ships, -- number of unique ships
            avg(duration_s) as avg_duration, -- average trajectory duration
            avg(st_length(track)) as avg_length, -- average length
            min(st_length(track)) as min_length -- shortest length
        FROM data_analysis.tracks
        GROUP BY start_cid, end_cid ) as q1
        LEFT JOIN data_analysis.clusters_stops_hulls as c1
            ON (c1.cid=q1.start_cid)
        LEFT JOIN data_analysis.clusters_stops_hulls as c2
            ON (c2.cid=q1.end_cid);
```

In order to improve the visualisation of the traffic statistics on a map, the lines of the graph edges can be bent using a custom *PL/pgSQL* function. Figure 12 illustrates the maritime traffic graph obtained using this function. The edges thickness is proportional to the number of trajectories associated to the track (i.e. thickest edges represent the most frequent routes between stationary areas). The navigation graph shown in Figure 12 is a simplified, summarised version of the density map shown in Figure 11.

5.4 Managing data quality using constraints

The previous sections highlight typical *PostgreSQL* and *PostGIS* features to process moving object data. The readers should be aware that the results of these queries may be affected by the quality of data. Apart from errors and irrelevant messages, AIS data in the given maritime dataset have been provided as received, including duplicates and other veracity issues. The coverage of the data is also not uniform, with 70.5% of vessel positions located in a range of 10 km from the AIS receiver [42]. Erroneous or missing AIS positions influence the quality of results but raise interesting algorithmic and processing challenges, therefore these veracity issues have been maintained in the dataset, which realistically represents the situation in the area and the period it refers to.

The accuracy of maritime clustering and trajectory tracks depend on data quality. For instance, the readers may compare the movement of passenger ships within the Brest roadstead versus the movement of passenger ships travelling to the Ushant and Molène islands where the data quality is lower. Pre-filtering, error-checking algo-

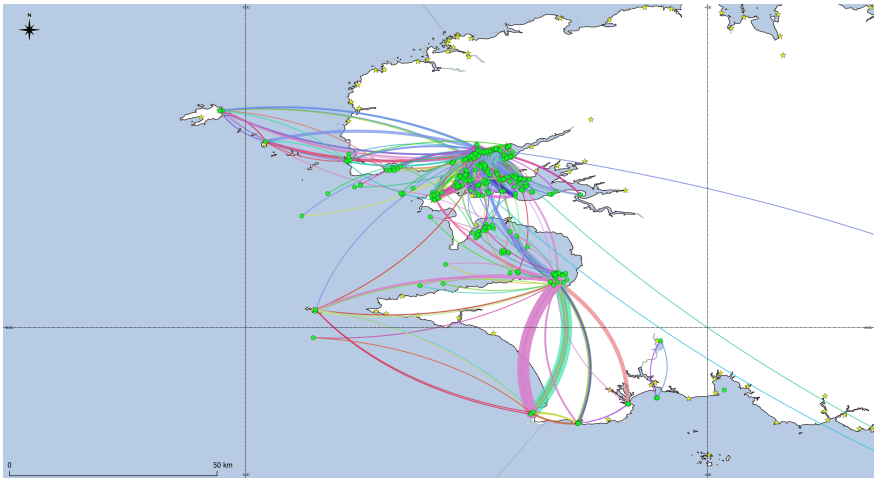

Fig. 12 Traffic graph. Graph nodes (green hexagons) represent stationary areas (e.g. ports, mooring areas), while edges (curved lines, one random colour per line) approximate tracks of vessels moving between stops.

rithms can affect the results as much as the variation of the analysis parameters (e.g. the speed threshold). Reasoning on intermediate structures can provide meaningful data quality information. For instance, the table `segments` can help detecting abnormal situations that can be ignored, like movements with extremely long duration, excessive distances between positions, or unfeasible speeds. Trajectories with unknown or erroneous MMSI identifiers could also be flagged as abnormal and discarded by the analysis.

In the database, *integrity constraints* may be used to manage data quality issues. Figure 13 presents two of the database schemas we built along the chapter, i.e. `ais_data` and `context_data`. The structure of the tables in these schemas is reported, with column names and types. Adding relations and constraints which are another essential part of database model design allows to manage data properties and to encompass the aforementioned quality issues in accordance with the database application. For example, integrity constraints may prevent that a ship appears travelling in two locations at the same time; may ensure that the vessel speed is always a positive value; and may guarantee that the vessel heading has either a default value (i.e. correctly setting it to 511), or is between 0 and 360 degrees.

Column constraints express integrity rules on the data in the specified columns. For instance, a `NOT NULL` constraint prevents the data in a column to assume the `NULL` value. *Table-level constraints* define additional rules that apply to all the data in a table. For instance, a `PRIMARY KEY` constraint combines a `NOT NULL` and a `UNIQUE` constraints, ensuring that all the data in a column (or in multiple columns, in combination) have unique values over the table. Figure 13 includes `PRIMARY KEY` constraints in both `ais_data` and `context_data` schema.

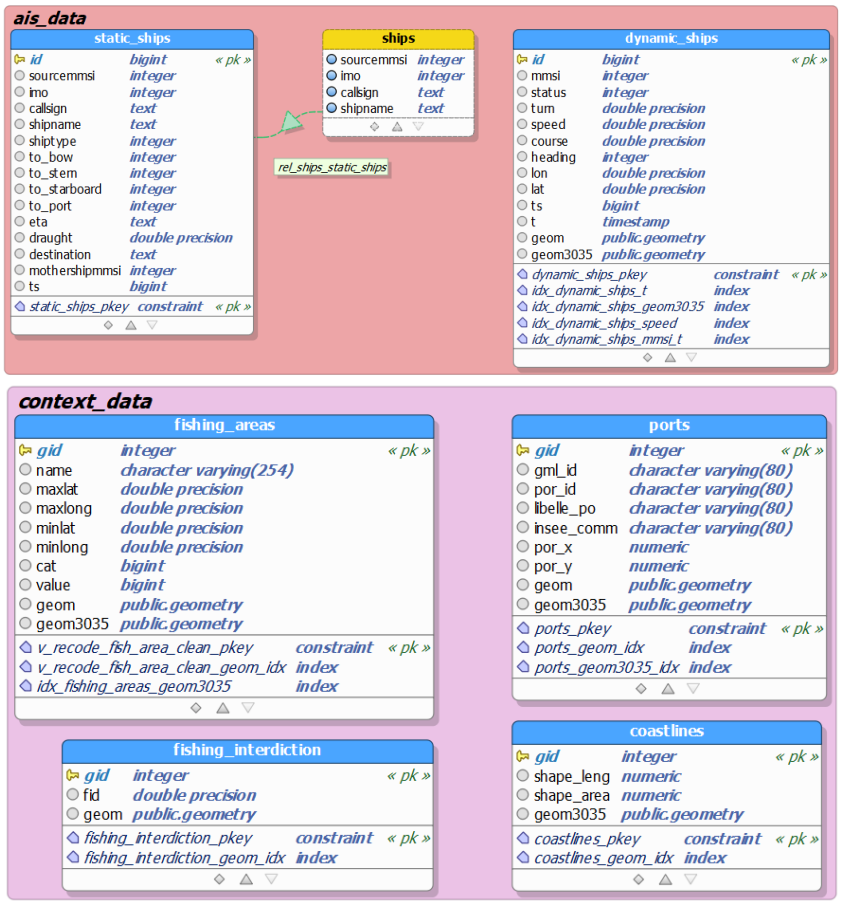

Fig. 13 Entities' structure (tables and views) of the database schemas `ais_data` and `context_data`. The data model is extracted from the *PostgreSQL* maritime database using pgModeler.[26]

Remark ! Modelling database constraints

Incorporating constraints in the database requires altering the tables' structure. Preferably, this modelling step should be accomplished altogether with the generation of the database, preceding the data insertion. Constraints can also be integrated along an iterative process, throughout the creation and the subsequent modification of the database.

Figure 14 presents the entities created to analyse the data along the chapter, and stored in the schema `data_analysis`. In the following, we are going to extend this part of the maritime database data model with constraints that preserve the data integrity when new operations (e.g. `INSERT`, `UPDATE`, `DELETE`) on tables and columns are executed.

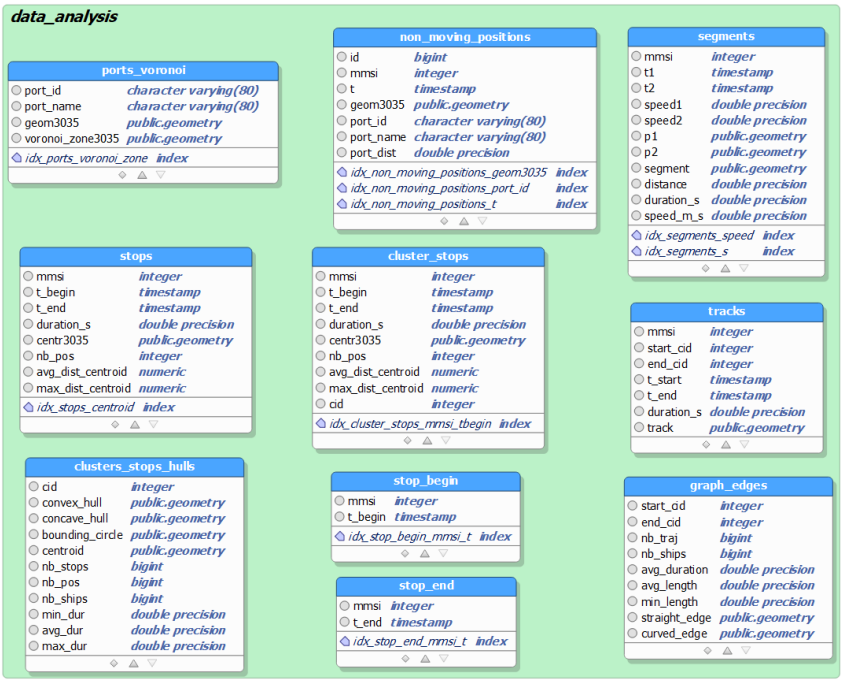

Fig. 14 Entities' structure (tables and views) of the database schema `data_analysis`. The data model is extracted from the *PostgreSQL* maritime database using pgModeler.

Cardinality constraints define how many entities may participate in a relationship. A *many-to-many* relationship is realised by a table that connects the entities in two other tables. *One-to-many* relationships are achieved with *foreign key constraints*, which maintain the referential integrity of data between two related tables. A `FOREIGN KEY` specifies that the values of the data in one or more columns must exist in a related table.

In the following example, a foreign key constraint is defined to ensure that all the ports for which a Voroni cell has been calculated, i.e. the ports in the table `data_analysis.ports_voronoi`, exist in table `context_data.ports`. In the example, after defining the tables' primary keys, a `FOREIGN KEY` constraint is specified altering the table `data_analysis.ports_voronoi`. This constraint realises a *one-to-many* relation between the two tables, as illustrated in Figure 15. The condition `ON DELETE CASCADE` in the SQL example avoids the creation of inconsistent, orphan Voronoi cells, which may be created when ports are removed from table `context_data.ports` but the corresponding Voronoi cell still exists. The constraint avoids this situation triggering the automatic deletion of the Voronoi cell that is associated to the deleted port.

SQL *Q5.4.1*

```
-- Table ports
ALTER TABLE context_data.ports
    DROP CONSTRAINT ports_pkey; -- Drop the old primary key which was
     automatically generated when importing the ports shapefile

ALTER TABLE context_data.ports
    ADD CONSTRAINT ports_pkey PRIMARY KEY (por_id); -- Column por_id is now
     the primary key of the table

-- Table ports_voronoi
ALTER TABLE data_analysis.ports_voronoi
    ADD CONSTRAINT ports_voronoi_pkey PRIMARY KEY (port_id); -- Column
     port_id is now the primary key of the table

ALTER TABLE data_analysis.ports_voronoi
    ADD CONSTRAINT ports_voronoi_fkey FOREIGN KEY (port_id)
    REFERENCES context_data.ports (por_id) ON DELETE CASCADE; -- Column
     port_id references the ports table using a foreign key.
```

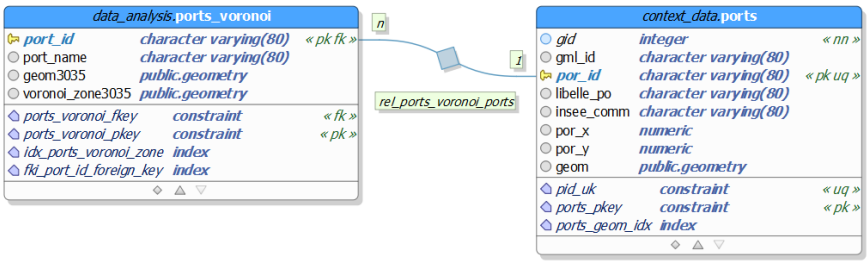

Fig. 15 Use of foreign keys to implement a one-to-many relation between ports and Voronoi tessellation (extracted from the *PostgreSQL* maritime database using pgModeler).

As a result of the activation of the foreign key constraint in the example, when creating a Voronoi cell for a port, this port must exist, i.e. all the ports in the table `data_analysis.ports_voronoi` must match some port in the referenced table `context_data.ports` (cf. keyword `REFERENCES`). However, the contrary may not hold, i.e. adding a new port without recomputing the Voronoi tessellation is allowed. In order to implement the dual constraint, a *one-to-one* relation must be defined. In this case, adding a new port would require generating the associated Voronoi cell.

Remark ! A note on *one-to-one* relations

One-to-one relations are not well represented in standard SQL, because they lead to a logical union of two tables. As a solution, reciprocal foreign keys may be used, i.e. each table in the relation has a foreign key reference to the primary key of the other table.This solution creates a circular dependency that may block the insertion of new data in the tables. Luckily, constraints may be deferred, i.e. lazily validated, allowing to temporarily violation of the integrity of the relation to enable the data insertion. The option `DEFERRABLE` can be used with this aim when defining the constraint.

As an alternative to the definition of reciprocal foreign keys, a *one-to-one* relation may be obtained by defining a *database trigger* to enforce it. This solution is illustrated in the following example. The trigger uses the database function `context_data.rebuild_table_ports_voronoi()`, which automatically generates a voronoi tessellation on the basis of the ports existing in table `context_data.ports`. The trigger `trigger_ports _voronoi` is defined for table `context_data.ports` to execute the function, rebuilding the Voronoi tessellation, whenever the table `context_data.ports` is modified (`AFTER INSERT OR UPDATE OR DELETE OR TRUNCATE on context_data.ports`).

Remark ! A note on triggers

Database triggers are SQL procedures that are executed upon the occurrence of a monitored database event, such as the insertion of a row in a table or view. In *PostgreSQL*, triggers are functions, which are automatically invoked by the DBMS whenever an insert, update, delete or truncate event occurs on a specified table.

SQL *Q5.4.2*

```
-- Create a function to rebuild the Voronoi tessellation
CREATE OR REPLACE FUNCTION context_data.rebuild_table_ports_voronoi()
    RETURNS TRIGGER AS $$ -- this function return a trigger
BEGIN
-- Same as Q5.1.1
DROP TABLE IF EXISTS data_analysis.ports_voronoi;
CREATE TABLE data_analysis.ports_voronoi AS
  SELECT por_id as port_id, libelle_po as port_name, geom3035,
    voronoi_zone3035
  FROM context_data.ports
  LEFT JOIN (
  SELECT (ST_Dump(ST_VoronoiPolygons(ST_Collect(geom3035)))).geom as
    voronoi_zone3035 FROM context_data.ports) as vp
  ON (ST_Within(ports.geom3035, vp.voronoi_zone3035));

-- Add primary and foreign key constraints
ALTER TABLE data_analysis.ports_voronoi ADD CONSTRAINT ports_voronoi_pkey
    PRIMARY KEY (port_id); -- primary key

ALTER TABLE data_analysis.ports_voronoi ADD CONSTRAINT ports_voronoi_fkey
    FOREIGN KEY (port_id) REFERENCES context_data.ports (por_id)
-- Column port_id references ports.por_id using a foreign key
    ON DELETE CASCADE; -- Deletes on ports are propagated

CREATE INDEX idx_ports_voronoi_zone ON data_analysis.ports_voronoi
USING gist (voronoi_zone3035);
  RETURN NEW; -- return the updated rows
END;
$$ LANGUAGE plpgsql; -- programming language used

-- Create a trigger to  execute the function whenever ports is updated
DROP TRIGGER IF EXISTS trigger_ports_voronoi ON context_data.ports;
CREATE TRIGGER trigger_ports_voronoi
    AFTER - the trigger is executed after a table update
    INSERT OR UPDATE OR DELETE OR TRUNCATE -- any type of update
    ON context_data.ports -- the monitored table
    FOR EACH STATEMENT -- after each statement
    EXECUTE PROCEDURE context_data.rebuild_table_ports_voronoi(); -- execute
      the function
```

The trigger can be tested by deleting a port from the database, as in the following example. As a result of the following SQL, the Voronoi table should be automatically updated.

SQL *Q5.4.3*

```
DELETE FROM context_data.ports WHERE libelle_po='Sein'; -- delete the port
    of the Sein Island
```

Action required

Study the database tables for identifying potential relations among them. Study the table columns, their types and expected values (cf. the files `README` in the maritime dataset) in order to define additional constraints using the aforementioned `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY` keywords. Consider also the `CHECK` constraint, which allows to evaluate and check a condition when inserting new data or updating existing ones, and `DEFAULT`, which permits to assign a default value when a new row is inserted in a table.

6 Summary and Conclusion

More than two decades ago, in an essay entitled *Marine Informatics: a new discipline emerges*, Roger Bradbury postulated that the scientific community was on the way to creating a new holistic discipline addressing the challenges of marine data integration and analysis [9]. One of the major issues he identified at that time was the lack of regularity and sparsity in the data collected, which were also temporally scattered. Nowadays, the large variety of maritime sensors together with scientific background and techniques for the monitoring, analysis, and visualisation of sea data has revolutionized the domain (*cf.* the chapter of Bereta et al. [7]). Positioning data correlated with contextual heterogeneous data as provided by the maritime dataset used in this chapter makes Bradbury's vision concrete, at least with respect to ships' movements.

This chapter, in line with Bradbury's vision, takes advantage of the navigation data brought by the maritime dataset to illustrate the benefits of relational database for *maritime informatics*. Specifically, the chapter addresses the design, storage and querying of maritime data through a spatio-temporal DBMS. In particular, the functionalities of a relational DBMS have been illustrated, due to the ability thereof to find, define, sort, modify, link and transform data in complex databases, while guaranteeing the user a robust layer for spatial analysis.

The open-source relational DBMS *PostgreSQL*, enhanced with the extension *PostGIS* for manipulating spatial features, was used to exemplify the concepts presented in the chapter. It is worth noticing that the proposed technical environment is open source, freely available and standard based, and is used in many academic and professional applications. As such, the reader may easily find additional material to extend the examples proposed herein. Furthermore, this technical choice is not to be considered as a limitation, because the examples of commands proposed in the chapter can find an easy correspondence in other DBMS that offer a spatial support.

The spatial representation, analysis and visualisation techniques presented in this chapter are a useful basis to understand the analytics approaches that are presented in the rest of the book, in particular in the chapters of Tampakis et al. [54] and of Andrienko et al. [2], which discuss data analytics and visual analytics, respectively.

7 Bibliographical Notes

The interested reader can refer to some additional material to get practical illustrations on the functionalities offered by spatio-temporal DBMS. The authors in [43] provide an introduction on the theoretical aspects of spatial databases, useful to better understand the different spatial object models and the available formats. An overview of the most important aspects of spatio-temporal databases is given in [27], which presents the main research results of the CHROROCRONOS project.[27] For the last research updates, the reader may refer to the series of the *International Symposium on Spatial & Temporal Databases (SSTD)*.

For additional material on OODBMS, the works extending the SECONDO DBMS offer many examples on how to extend an OODBM model to support complex spatio-temporal queries like group spatio-temporal patterns [47] and range-queries [58]. [16] shows also an extension of the SECONDO model to support symbolic trajectories. The reader interested in distributed computation can have a look to Parallel SECONDO ,[28] developed to improve the performance of mobility data analysis that supports a specialised version of the *R-Tree* index called *TM-RTree*.

Practical examples of the combined used of *PostgreSQL*/*PostGIS* and *QGIS* for movement data analysis are available in Anita Graser's blog.[29] We also refer to the official *PostgreSQLPostGIS* documentation for a detailed description of spatial and temporal database functionalities. Note that, although *PostgreSQL* and *PostGIS* are mainly used for two dimensions geometries (X and Y coordinates), *PostGIS* also supports the handling of three-dimensional (3-D) geometries. This extra dimension, namely Z, is added to each vertex in the geometry, and the geometry type itself is enhanced accordingly, to enable the correct interpretation of the additional dimension. For instance, the 2-D geometries `Point`, `Linestring` and `Polygon` become 3-D `PointZ`, `LinestringZ` and `PolygonZ`, respectively. Also, the use of the 3-D only `Polyhedralsurface` makes possible the generation of volumetric objects in the database. Special *PostGIS* functions for spatial relationships have been adapted to 3-D geometries. The reader can refer to the official *PostGIS* documentation.[30]

---

[27] CHROROCRONOS project `http://chorochronos.datastories.org`

[28] Parallel SECONDO `http://dna.fernuni-hagen.de/secondo/ParallelSecondo/`

[29] Anita Graser's blog `https://anitagraser.com/`

[30] *PostGIS* 3-D `https://postgis.net/workshops/postgis-intro/3d.html`

The NoSQL Database management website[31] provides a useful overview of the capabilities of existing NoSQL systems, including spatial extensions. Recently, [6] described a proposal to extend column-based *Cassandra* stores to the spatial dimension. Another solution to spatio-temporal objects persistence for large-scale data and geo-spatial analytics is *GeoMesa*,[32] an open-source suite of tools that interfaces with NoSQL databases like *Google Bigtable* and *Cassandra*, among others. *GeoMesa* supports near-real time analytics for streaming data and distributed data processing, and relies on *GeoServer*,[33] a well known data server for geographic data, and OGC application programming interfaces for map server integration.

Port and stationary areas detection, which is discussed in the chapter, is a hot topic in maritime related research. Most of the proposed approaches use unsupervised learning. For instance, Millefiori *et al.* use a data-driven approach (i.e. kernel density estimation (KDE) on AIS data) to define the extended areas of operation of seaports [34]. Similarly, Vespe *et al.* [56] also use density estimates on AIS data to map fishing activities at European scale. This topic is also addressed in the chapter of Andrienko et al. [2] in this book. Taking advantage of the identified stationary areas, vessel trajectory analysis and prediction [13, 59], identification of human activities at sea [49], anomaly detection [45, 31, 30, 51] are also widely developed in the literature.

The reader interested in uncertainty representation and reasoning in maritime data and information fusion can also refer to [15, 46, 5] and the chapter of Jousselme and Pallotta [23].

References

1. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.
2. Natalia Andrienko and Gennady Andrienko. Visual analytics for maritime studies. In A. Artikis and D. Zissis, editors, *Maritime Informatics*, chapter 5. Springer, 2020.
3. Natalia V. Andrienko and Gennady L. Andrienko. Spatial generalization and aggregation of massive movement data. *IEEE Transactions on Visualization and Computer Graphics*, 17:205–219, 2011.
4. Franz Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.
5. Giulia Battistello and Wolfgang Koch. Knowledge-aided multi-sensor data processing for maritime surveillance. In *GI Jahrestagung (2)*, pages 796–799, 2010.
6. Mohamed Ben Brahim, Wassim Drira, Fethi Filali, and Noureddine Hamdi. Spatial data extension for cassandra nosql database. *Journal of Big Data*, 3(1):11, Jun 2016.
7. Kostantina Bereta, Konstantinos Chatzikokolakis, and Dimitris Zissis. Maritime reporting systems. In A. Artikis and D. Zissis, editors, *Maritime Informatics*, chapter 1. Springer, 2020.
8. Piotr Borkowski. The ship movement trajectory prediction algorithm using navigational data fusion. *Sensors*, 17(6):1432, 2017.

[31] NoSQL Database management website `http://nosql-database.org/`

[32] GeoMesa `https://www.geomesa.org/`

[33] Geoserver `http://geoserver.org/`

9. Roger Bradbury. Marine informatics: a new discipline emerges. *Maritime Studies*, 1995(80):15–22, 1995.
10. Emanuele Carlini, Vincius Monteiro, Amilcar Soares, Mohammad Etemad, Bruno Machado, and Stan Matwin. Uncovering vessel movement patterns from ais data with graph evolution analysis. In *Proceedings of the MASTER workshop, 23rd International Conference on Extending Database Technology (EDBT)*, pages 1–7, 2020.
11. Max Egenhofer. A mathematical framework for the definition of topological relations. In *Proc. the fourth international symposium on spatial data handing*, pages 803–813, 1990.
12. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.
13. Shaojun Gan, Shan Liang, Kang Li, Jing Deng, and Tingli Cheng. Ship trajectory prediction for intelligent traffic management using clustering and ann. In *Control (CONTROL), 2016 UKACC 11th International Conference on*, pages 1–6. IEEE, 2016.
14. Raffaele Grasso. Ship classification from multi-spectral satellite imaging by convolutional neural networks. In *Proc. of the 27th European Signal Processing Conference, A Curuna, Spain, September 2-6*, 2019.
15. Marco Guerriero, Peter Willett, Stefano Coraluppi, and Craig Carthel. Radar/ais data fusion and sar tasking for maritime surveillance. In *Information Fusion, 2008 11th International Conference on*, pages 1–5. IEEE, 2008.
16. Ralf Hartmut Güting, Fabio Valdés, and Maria Luisa Damiani. Symbolic trajectories. *ACM Trans. Spatial Algorithms Syst.*, 1(2):7:1–7:51, July 2015.
17. Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.
18. Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 562–573, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
19. Bo Huang and Christophe Claramunt. STOQL: An ODMG-based spatio-temporal object model and query language. In Dianne E. Richardson and Peter van Oosterom, editors, *Advances in Spatial Data Handling*, pages 225–237, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
20. ISO Central Secretary. Geographic information. simple feature access. sql option. Standard ISO 19125-2:2006, International Organization for Standardization, Geneva, CH, 2006.
21. Alya Itani, Cyril Ray, Ammar El Falou, and John Issa. Mining Ship Motions and Patterns of Life for the EU Common Information Sharing Environment (CISE). In *OCEANS 2019, Marseille, France*, pages 1–6, 2019.
22. Tor A Johansen, Andrea Cristofaro, and Tristan Perez. Ship collision avoidance using scenario-based model predictive control. *IFAC-PapersOnLine*, 49(23):14–21, 2016.
23. Anne-Laure Jousselme and Giuliana Pallotta. Dissecting uncertainty handling techniques for maritime anomaly detection. In A. Artikis and D. Zissis, editors, *Maritime Informatics*, chapter 8. Springer, 2020.
24. Alen Jugović, Svjetlana Hess, and Tanja Poletan Jugović. Traffic demand forecasting for port services. *Promet-Traffic&Transportation*, 23(1):59–69, 2011.
25. Sungho Kim and Joohyoung Lee. Small infrared target detection by region-adaptive clutter rejection for sea-based infrared search and track. *Sensors (Basel)*, 14:13210–13242, 2014.
26. Ravi Kothuri and Siva Ravada. *Oracle Spatial, Geometries*, pages 821–826. Springer US, Boston, MA, 2008.
27. M. Koubarakis, T. Sellis, A.U. Frank, S. Grumbach, R.H. Gting, C.S. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, editors. *Spatio-Temporal Databases - The CHOROCHRONOS Approach*, volume 2520 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg, 2003.

28. Ahmet Kucuk, Shah Muhammad Hamdi, Berkay Aydin, Michael A. Schuh, and Rafal A. Angryk. Pg-trajectory: A postgresql/postgis based data model for spatiotemporal trajectories. In *Proceedings of the 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), 8-10 October 2016, Atlanta, GA, USA*, pages 81–88, 2016.
29. Wissame Laddada and Cyril Ray. Graph-based analysis of maritime patterns of life. In *Proceedings of the GAST Workshop, 20th Journées Francophones Extraction et Gestion des Connaissances (EGC)*, pages 1–14, 2020.
30. Richard O Lane, David A Nevell, Steven D Hayward, and Thomas W Beaney. Maritime anomaly detection and threat assessment. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–8. IEEE, 2010.
31. Bo Liu, Erico N de Souza, Cassey Hilliard, and Stan Matwin. Ship movement anomaly detection using specialized distance measures. In *Information Fusion (Fusion), 2015 18th International Conference on*, pages 1113–1120. IEEE, 2015.
32. Antonios Makris, Konstantinos Tserpes, Giannis Spiliopoulos, and Dimosthenis Anagnostopoulos. Performance Evaluation of MongoDB and PostgreSQL for Spatio-temporal Data. In *2nd International Workshop on Big Mobility Data Analytics (BMDA2019), Lisbon, Portugal*, March 2019.
33. Fabio Mazzarella, Alfredo Alessandrini, Harm Greidanus, Marlene Alvarez, Pietro Argentieri, Domenico Nappo, and Lukasz Ziemba. Data fusion for wide-area maritime surveillance. In *Proceedings of the COST MOVE Workshop on Moving Objects at Sea, Brest, France*, pages 27–28, 2013.
34. Leonardo M Millefiori, Dimitrios Zissis, Luca Cazzanti, and Gianfranco Arcieri. Scalable and distributed sea port operational areas estimation from ais data. In *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on*, pages 374–381. IEEE, 2016.
35. International Standard Organisation. Iso/iec 9075:2016 information technology database languages sql.
36. Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady Andrienko, Natalia Andrienko, Vania Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, Jose Macedo, Nikos Pelekis, et al. Semantic trajectories modeling and analysis. *ACM Computing Surveys (CSUR)*, 45(4):42, 2013.
37. Kostas Patroumpas. Online mobility tracking against evolving maritime trajectories. In A. Artikis and D. Zissis, editors, *Maritime Informatics*, chapter 6. Springer, 2020.
38. Nikos Pelekis, Yannis Theodoridis, Spyros Vosinakis, and Themis Panayiotopoulos. Hermes - A framework for location-based data management. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, pages 1130–1134, 2006.
39. Gohar A Petrossian. Preventing illegal, unreported and unregulated (iuu) fishing: A situational approach. *Biological Conservation*, 189:39–48, 2015.
40. David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann Publishers Inc., 1992.
41. Cyril Ray, Richard Dréo, Elena Camossi, Anne-Laure Jousselme, and Clément Iphar. Heterogeneous integrated dataset for maritime intelligence, surveillance, and reconnaissance. *Data in Brief*, Vol. 25:104141, 2019.
42. Cyril Ray, Richard Dréo, Elena Camossi, Anne-Laure Jousselme, and Clément Iphar. Heterogeneous integrated dataset for maritime intelligence, surveillance, and reconnaissance. *Data In Brief*, 25, 2019.
43. Philippe Rigaux, Michel Scholl, and Agnès Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
44. Maria Riveiro, Giulian Pallotta, and Michele Vespe. Maritime anomaly detection: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 0(0):e1266, 2018.
45. Stephen Roberts. *Anomaly detection in vessel track data*. PhD thesis, Oxford University, UK, 2014.

46. Jean Roy and Eloi Bosse. Sensor integration, management and data fusion concepts in a naval command and control perspective. Technical report, Defence Research Establishment Valcartier (Québec), 1998.
47. Mahmoud Attia Sakr and Ralf Hartmut Güting. Group spatiotemporal pattern queries. *Geoinformatica*, 18(4):699–746, October 2014.
48. W. Siabato, MA. Manso-Callejo, and E. Camossi. An annotated bibliography on spatiotemporal modelling trends. *International Journal of Earth and Environmental Sciences*, 2(135):26 pp., 2017.
49. PAM Silveira, AP Teixeira, and C Guedes Soares. Use of ais data to characterise marine traffic patterns and ship collision risk off the coast of portugal. *The Journal of Navigation*, 66(6):879–898, 2013.
50. Rolf Simoes, Gilberto Queiroz, Karine Ferreira, Lubia Vinhas, and Gilberto Cmara. Postgist: towards a spatiotemporal postgresql database extension. In *XVII Brazilian Symposium on Geoinformatics (GeoInfo 2016)*, 2016.
51. Behrouz Haji Soleimani, Erico N De Souza, Casey Hilliard, and Stan Matwin. Anomaly detection in maritime data based on geometrical analysis of trajectories. In *Information Fusion (Fusion), 2015 18th International Conference on*, pages 1100–1105. IEEE, 2015.
52. Stefano Spaccapietra, Christine Parent, Maria Luisa Damiani, Jose Antonio de Macedo, Fabio Porto, and Christelle Vangenot. A conceptual view on trajectories. *Data & Knowledge Engineering*, 65(1):126 – 146, 2008. Including Special Section: Privacy Aspects of Data Mining Workshop (2006) - Five invited and extended papers.
53. Christian Strobl. *PostGIS*, pages 891–898. Springer US, Boston, MA, 2008.
54. Panagiotis Tampakis, Stylianos Sideridis, Panagiotis Nikitopoulos, Nikos Pelekis, and Yannis Theodoridis. Maritime data analytics. In A. Artikis and D. Zissis, editors, *Maritime Informatics*, chapter 4. Springer, 2020.
55. Alejandro Vaisman and Esteban Zimányi. Mobility data warehouses. *ISPRS International Journal of Geo-Information*, 8(4), 2019.
56. Michele Vespe, Maurizio Gibin, Alfredo Alessandrini, Fabrizio Natale, Fabio Mazzarella, and Giacomo C Osio. Mapping eu fishing activities using ship tracking data. *Journal of Maps*, 12(sup1):520–525, 2016.
57. Jianqiu Xu and Ralf Hartmut Güting. A generic data model for moving objects. *Geoinformatica*, 17(1):125–172, January 2013.
58. Jianqiu Xu, Hua Lu, and Ralf Hartmut Guting. Range queries on multi-attribute trajectories. *IEEE Trans. on Knowl. and Data Eng.*, 30(6):1206–1211, June 2018.
59. Tingting Xu, Xiaoming Liu, and Xin Yang. Ship trajectory online prediction based on bp neural network algorithm. In *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, volume 1, pages 103–106. IEEE, 2011.
60. Hongchu Yu, Zhixiang Fang, Feng Lu, Alan T. Murray, Zhiyuan Zhao, Yang Xu, and Xiping Yang. Massive automatic identification system sensor trajectory data-based multi-layer linkage network dynamics of maritime transport along 21st-century maritime silk road. *Sensors*, 19(19), 2019.