ULB UNIVERSITÉ LIBRE DE BRUXELLES

Java Database Connectivity Driver for MobilityDB

Master Thesis

Tatiana del Pilar Millan Poveda

Master thesis submitted under the supervision of
Prof. Esteban Zimanyi

Academic year
2021 - 2022

In order to be awarded the Master's programme in
Computer Science

Abstract

The purpose of this project is the creation of a *Java Database Connectivity Driver (JDBC)* that helps programs written in Java to connect with MobilityDB databases by extending PostgreSQL JDBC types. MobilityDB is an open-source database management system extension of PostgreSQL, that offers support for object moving data and special temporal data.
The *MobilityDB-JDBC* driver provides support for the new MobilityDB types: Period, PeriodSet, TimestampSet, TBox, STBox , TInt, TBool, TFloat, TText, TGeomPoint and TGeogPoint and also the temporal types TInstant, TInstantSet, TSequence and TSequenceSet, and was built based on the MobilityDB Python adapter with the objective of acquire feature parity.

Keywords: MobilityDB, Java, JDBC, driver, PostgreSQL, PostGIS

Contents

1 Introduction **1**

2 Technologies **2**

2.1 PostgreSQL 2

2.2 PostGIS 3

2.3 MobilityDB 4

2.3.1 Python adapter 7

2.4 JDBC driver 9

2.4.1 PostgreSQL JDBC 9

2.4.2 PostGIS JDBC 9

2.5 Javadocs 10

2.6 Unit Tests and Integration Tests 11

2.7 JUnit 11

2.8 SonarQube 12

3 Design **13**

3.1 Code Validation 15

3.1.1 Unit Tests 15

3.1.2 Integration Tests 16

3.2 Example 17

4 Implementation **18**

4.0.1 Shared Concepts 18

4.1 Time Types 19

4.1.1 TimestampSet 19

4.1.1.1 Constructors 19

4.1.1.2 Methods 20

4.1.2 Period 20

4.1.2.1 Constructors 21

4.1.2.2 Properties 21

4.1.2.3 Methods 21

4.1.3 PeriodSet . . . 22
4.1.3.1 Constructors . . . 22
4.1.3.2 Methods . . . 22
4.2 Box Types . . . 23
4.2.1 TBox . . . 24
4.2.1.1 Constructors . . . 24
4.2.1.2 Properties . . . 25
4.2.2 STBox . . . 25
4.2.2.1 Constructors . . . 25
4.2.2.2 Properties . . . 28
4.3 Temporal Types . . . 29
4.3.0.1 Shared concepts . . . 31
4.3.0.2 TBool . . . 31
4.3.0.3 TFloat . . . 32
4.3.0.4 TInt . . . 32
4.3.0.5 TGeomPoint . . . 33
4.3.0.6 TGeogPoint . . . 34
4.3.0.7 TText . . . 35
4.3.1 Temporal Instant . . . 37
4.3.1.1 TBoolInst . . . 37
4.3.1.2 TFloatInst . . . 38
4.3.1.3 TIntInst . . . 38
4.3.1.4 TGeomPointInst . . . 38
4.3.1.5 TGeogPointInst . . . 39
4.3.1.6 TTextInst . . . 39
4.3.2 Temporal InstantSet . . . 40
4.3.2.1 TBoolInstSet . . . 40
4.3.2.2 TFloatInstSet . . . 41
4.3.2.3 TIntInstSet . . . 42
4.3.2.4 TGeomPointInstSet . . . 42
4.3.2.5 TGeogPointInstSet . . . 44
4.3.2.6 TTextInstSet . . . 45
4.3.3 Temporal Sequence . . . 46
4.3.3.1 TBoolSeq . . . 46
4.3.3.2 TFloatSeq . . . 48
4.3.3.3 TIntSeq . . . 50
4.3.3.4 TGeomPointSeq . . . 51
4.3.3.5 TGeogPointSeq . . . 56
4.3.3.6 TTextSeq . . . 61

4.3.4 Temporal SequenceSet 62
4.3.4.1 TBoolSeqSet 63
4.3.4.2 TFloatSeqSet 63
4.3.4.3 TIntSeqSet 65
4.3.4.4 TGeomPointSeqSet 65
4.3.4.5 TGeogPointSeqSet 68
4.3.4.6 TTextSeqSet 70

5 Development Environment **72**
5.1 Requirements 72
5.2 Build 72
5.3 Code Analysis 73
5.3.1 SonarQube Docker Image 73
5.4 Running Integration Tests 74
5.5 MobilityDB Docker Image 74

6 Running the Example **75**
6.1 Preparation 78
6.2 Execution 78
6.3 Drawing the Results in QGIS 79

7 Conclusions **82**

Appendices **83**

A Example of PostGIS JDBC Usage **84**

B Methods by DataType **86**

C Methods Definition **88**

List of Tables

2.1 Advantages of PostgreSQL vs. Firebird and MariaDB 3
2.2 Spatial Functions Categories [11] 4
2.3 MobilityDB requirements [5] 7
2.4 MobilityDB Python adapter requirements[45] 7
2.5 Temporal types subclasses 8

6.1 Comparison of times when runing in Linux virtual machine and in Windows 78
6.2 Number of records by table 78

B.1 Methods by data type - Time and Box types 86
B.2 Methods by data type - Temporal types 87

C.1 Methods definition 89

List of Figures

2.1 Spatial data types are organized in a type hierarchy. [11] 4
2.2 MobilityDB stack [46] 5
2.3 MobilityDB architecture in detail[46] 6
2.4 Python Classes: UML Diagram[46] 8
2.5 Registering data type to the PostgreSQL connection [10] 9
2.6 Retrieving Geometry from PostgreSQL database [10] 10
2.7 Generated documentation with Javadoc tool 10
2.8 Comparison between Unit Testing and Integration Testing [38] 11
2.9 SonarQube functionality example[34] 12

3.1 PGObject class [31] 13
3.2 MobilityDB JDBC general diagram 14
3.3 SonarQube analysis for MobilityDB-JDBC 15

4.1 Time type architecture diagram 19
4.2 Box type architecture diagram 23
4.3 Temporal type architecture diagram 30
4.4 TInstant architecture diagram 37
4.5 TInstantSet architecture diagram 40
4.6 TSequence architecture diagram 46
4.7 TSequenceSet architecture diagram 62

6.1 QGis: Load the map 79
6.2 QGis: Create the connection to PostgreSQL 80
6.3 QGis: Select the table from the database 80
6.4 QGis: Trajectories drawn in the map. 81

A.1 Usage of PostGIS JDBC [10] 85

Chapter 1

Introduction

MobilityDB is an open-source management system for moving object geospatial trajectories, developed by the Computer & Decision Engineering Department of the Université Libre de Bruxelles [5].

Moving objects change values or position within the time, normally used for location tracking. MobilityDB stores and retrieves moving object data, adding functions and operators to query this data efficiently [17]. In order to use it in a Python environment, the MobilityDB team has developed a Python driver and an extension to SQLAlchemy [27] for the interaction with MovingPandas[20], Flask and MobilityDB.

Nowadays the most used languages around the world are Python, C, Java respectevely, according to the Tiobe ranking [8]. This ranking is calculated over the search of in the most popular web browsers, the skilled engineers, courses and third-party vendors. It is updated each month and Java is always getting the first places, but this is not the only ranking where Java has a good place, in the ranking of PYPL PopularitY of Programming Language [9] based on Google trends, Java is also getting the first places.

We can say that as Java is a world-wide popular language, and it has a very big community; therefore having a Java driver to connect MobilityDB would make it easier for the interested people to give a try and start using it.

The main objective of the Master Thesis is to create a Java Database Conectivity Driver for MobilityDB, to extend its use and make it easier to access and integrate from one of the most worldwide used programming languages, creating feature parity with the already developed Python adapter.

Chapter 2

Technologies

2.1 PostgreSQL

PostgreSQL is an open-source object-relational highly stable database [23], that is extensible and ACID-compliant(Atomicity, Consistency, Isolation and Durability) assuring that a database transaction is processed in a consistently and safe way. It supports SQL standards and provides features as complex queries, triggers, foreign keys, multi-version concurrency control, etc.[22]. The integration PostgreSQL supported languages [40] are: Python, Java, C, C++, C#, Ruby, Javascript, etc.

One of the main interesting characteristics of PostgreSQL is the extensibility: users can extend data types, function operators, index methods, aggregated functions, etc.. There are other important features [40] as:

- Table inheritance
- Sophisticated locking mechanism
- Foreign key referential integrity
- Views, rules, subquery
- Nested transactions (savepoints)
- Multi-version concurrency control (MVCC)
- Asynchronous replication
- Native Microsoft Windows Server version
- Tablespaces
- Point-in-time recovery

In comparison to other open-source relational databases like Firebird and MariaDB, PostgreSQL offers some advantages[35], described in the Table 2.1

Feature	***Description***
Data model	Supports user defined objects, data types, functions, operators, domains and indexes
Data types and structures	Uuid, monetary, enumerated, geometric, binary, network address, bit string, text search, xml, json, array, composite and range types, internal types for object identification and log location.
Network addresses	Network address types CIDR ,INET, MACADDR (for MAC addresses)
Multi-dimensional arrays	Array size can be specified for most of the data types
Geometric data	Data types such as points, lines, circles, and polygons. Path data type sequence of points
JSON support	JSON, JSONB(binary form) datatypes to use JSON operators and functions
Create a new type range and base.	Create new data types as composite, enumerated,
Data size	Unlimited database size, 32TB table size.
Data integrity	ACID-compliant, supports 160 out of 179 features for full core SQL:2011 compliance. [15]

Table 2.1: Advantages of PostgreSQL vs. Firebird and MariaDB

On the other hand, some disadvantages are the performance, the memory allocation and creation of separate services for each client, the speed in comparison with for example with MySQL, the replication and more advanced setup[16].

2.2 PostGIS

PostGIS is a spatial database extender for PostgreSQL [11], it extends the PostgreSQL features, giving geospatial types and functions to improve the managing and querying of spatial data in a language similar to language[2].

PostGIS augments the existing geometric data features with additional spatial types, functions, operators, and indexes [35]. It also provides integration with geospatial tools to map and render data.

The three important aspects of spatial data, to obtain an optimized and flexible structure [11]:

1. **Spatial data types** to represent geographic features. They are considered shapes, they have a hierarchy shown in figure 2.1.

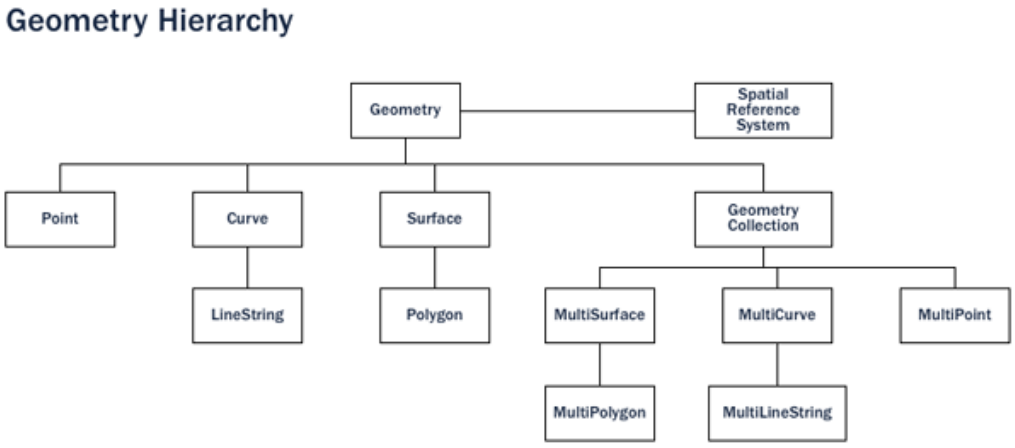

Figure 2.1: Spatial data types are organized in a type hierarchy. [11]

2. **Multidimensional spatial indexing** for processing of spatial operations in an efficient way. The multidimensional index used by PostGIS is R-Tree, that divides the data in regions adjusting the boundaries, usually rectangles trying to avoid overlapping, a common issue with multidimensional indexes.

3. **Spatial functions** to manipulate geometries, analyze spatial relationships and components. They are used for doing queries over spatial data. They can be grouped in the five categories:

Category	***Description***
Conversion	Convert between geometries and external data formats.
Management	Manage information about spatial tables and PostGIS administration.
Retrieval	Retrieve properties and measurements of a Geometry.
Comparison	Compare two geometries with respect to their spatial relation.
Generation	Generate new geometries from others.

Table 2.2: Spatial Functions Categories [11]

Some advantages of using PostGIS are:

- Easy spatial querying and manipulation of data
- Integration with other tools and third-parties

2.3 MobilityDB

MobilityDB is a database management system that adds support for moving object data, special temporal data to PostgreSQL and its PostGIS extension[28] [5]. The

moving object data are objects that move over time and change is geographical location. They are tracked and that information is stored for managing and analysis.

MobilityDB is open-source and compliant with OGC standards on Moving Features, especially in Moving Access. It is developed by the Computer & Decision Engineering Department of the Université Libre de Bruxelles (ULB).

The way the information is stored is with a spatial point 2D or 3D that has its corresponding timestamp. A trajectory is a set of connected points. MobilityDB provides special types as temporal points, floats, booleans, int, etc. to do calculations over the temporal types. There are also instants that represent a single location with its timestamp.

There are like 2300 operations over the temporal types [17], classified in three categories [44]:

- Lifed functions and operators: MobilityDB takes care of the temporal aspects and delegates the spatial processing to PostGIS. [44], these operations are performed on base types and make sense for temporal types [17].
- Temporal functions and operators: process the temporal dimension of the value an instant, a range, an array of instant, or an array of ranges [44], only make sense for temporal arguments [17].
- Spatiotemporal functions and operators: the non-grouped functions are considered here like velocity, azimuth, maxValue,etc.

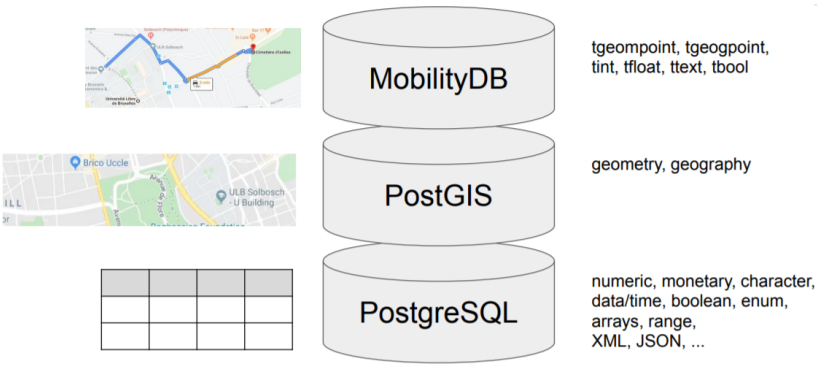

Figure 2.2: MobilityDB stack [46]

Figure 2.2 shows the stack order and the position of MobilityDB in respect to PostgreSQL and PostGIS. At the bottom of the stack is PostgreSQL, that supports relational types, the PostGIS with geometry and geography abstractions for coordinates in maps is in the middle and in top of them MobilityDB with its temporal values to represent movements over time [46].

The main function of MobilityDB is storing and querying trajectories efficiently [28], while PostGIS creates a trajectory in a query at application level, MobilityDB will do it at the database level [46].

The architecture of MobilityDB is a vertical extension of PostgreSQL,it starts with the data model by extending spatiotemporal types for representing moving points or moving network points. The Access methods supports GIST, SP-GIST, B-tree to support queries on large tables, the query optimizer is extended to add functionality for analyzing temporal types, a big set of operations like distance, aggregations, predicates, ETL, etc. They are packaged and can be used through the Python diver and by third-party tools, maintaining the compatilibility SQL ecosystem [46]. Figure 2.3 shows the components in detail.

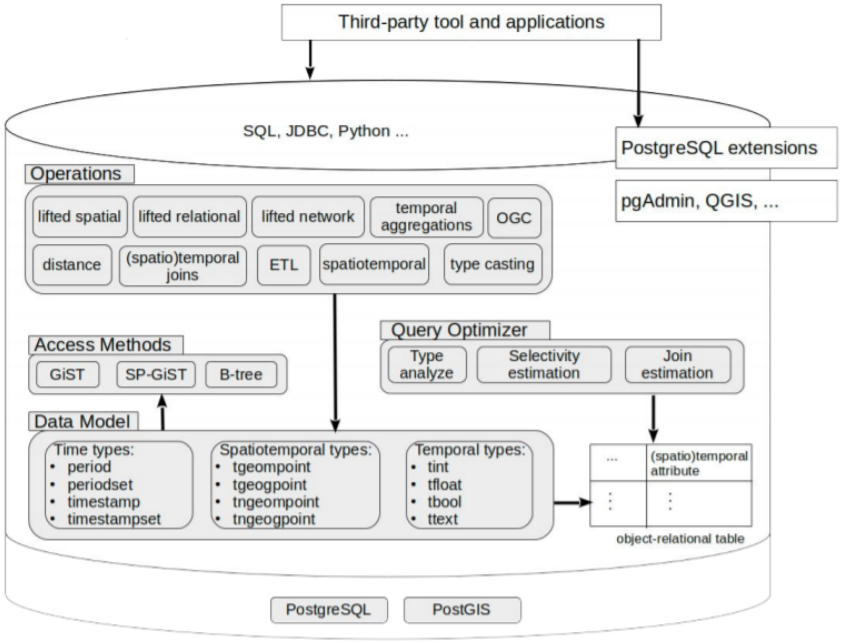

Figure 2.3: MobilityDB architecture in detail[46]

For having big spatiotemporal management, MobilityDB uses Citus [13], an extension that horizontally scales PostgreSQL across multiple machines using sharding and replication. One coordinator node and n worker nodes, with the same stack PostgreSQL, PostGIS, MobilityDB and Citus, some intial configuration for distribution of data.

Requirement	***Version***
PostgreSQL	>10
CMake	>= 3.1
PostGIS	== 2.5
JSON-C	
GNU Scientific Library (GSL)	
Development files for PostgreSQL, PostGIS/liblwgeom, PROJ & JSONC	

Table 2.3: MobilityDB requirements [5]

.

MobilityDB was tested on Linux operating systems, the requirements for MobilityDB are shown in the table 2.3.

2.3.1 Python adapter

The Python adapter for MobilityDB python-mobilitydb [45], supports the adapters psycopg2[18] and the asyncpg [3] for PostgreSQL and python-postgis [39] for PostGIS.

Requirement	***Version***	***Description***
Python	3.x	
psycopg2 or asyncpg		Connection to PostgreSQL
Spans		PostgreSQL's range types
postgis		Connection to PostGIS
python-dateutil		datetime extensions
parsec		Parser library

Table 2.4: MobilityDB Python adapter requirements[45]

.

The requirements for using this adapter are listed on the table 2.4.

This adapter acts as a translator to the temporal types or from them. The temporal types are TBool (temporal booleans), TInt (temporal integers), TText (temporal strings), TFloat(temporal floats), TGeomPoint(temporal geometric points) and TGeogPoint(geographic points with spherical coordinates) [45].

Temporal Type	***Subclasses***
TBool	TBoolInst, TBoolInstSet, TBoolSeq, and TBoolSeqSet.
TInt	TIntInst, TIntInstSet, TIntSeq, and TIntSeqSet.
TText	TTextInst, TTextInstSet, TTextSeq, and TTextSeqSet.
TFloat	TFloatInst, TFloatInstSet, TFloatSeq, and TFloatSeqSet.
TGeomPoint	TGeomPointInst, TGeomPointInstSet, TGeomPointSeq, and TGeomPointSeqSet.
TGeogPoint	TGeogPointInst, TGeogPointInstSet, TGeogPointSeq, and TGeogPointSeqSet.

Table 2.5: Temporal types subclasses

.

Each of the types has its subclasses defined by the following class generalization shown in figure 2.4

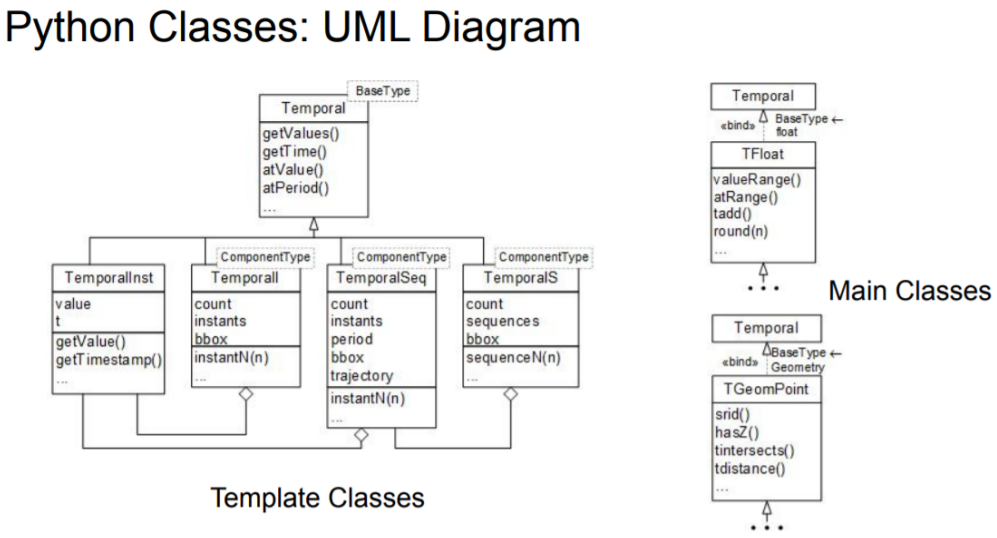

Figure 2.4: Python Classes: UML Diagram[46]

The generic type Temporal accepts some base type like integer, float, boolean, etc; there are four subtypes [46]:

- Instant, to represent a single instant
- Instant set, to represent a set of instants (without interpolation)
- Sequence, to represent trajectories, continues movement (with interpolation)
- Sequence set, to introduce temporal gaps in the movement

2.4 JDBC driver

PostgreSQL JDBC Driver gives the ability to communicate Java programs with PostgreSQL databases using a network protocol[21] Supported authentication methods by the JDBC driver are the trust, ident, password, md5, gss and crypt authentication configured in the *pg_hba.conf* file [24].

Extensibility is one of the main features in PostgreSQL, creating own functions and own data types, to handle this, the is a set of extension API's. The class *org.postgresql.PGConnection* has methods to support extension [24].

JDBC driver is extended to support PostGIS data types (Geometry, Box2D, Box3D) and parse to the corresponding data type, instead of receiving byte arrays or strings. This extension needs to be registered to get as result in the expected data type before querying the database [12].

2.4.1 PostgreSQL JDBC

The PgJDBC is an open-source Java driver for connecting PostgreSQL databases with code written in Java [33].

The PostgreSQL JDBC *addDataType* method allows us to create a custom data type, that is new to the driver, just specifying the type and the class name [32].

```
public void addDataType(String type, String name)
```

The new class should inherit from *PGObject* (org.postgresql.util.PGobject)[32].

```
...
((org.postgresql.PGConnection)myconn).addDataType("mytype",
   "my.class.name");
 ...
```

2.4.2 PostGIS JDBC

The Geometry objects of PostGIS in a PostgresQL database can be accessed from Java code using the JDBC extension or directly [10].

The most important parts are registering the type to the connection:

```
/*
* Add the geometry types to the connection. Note that you
* must cast the connection to the pgsql-specific connection
* implementation before calling the addDataType() method.
*/
((org.postgresql.PGConnection)conn).addDataType("geometry",Class.forName("org.postgis.PGgeometry"));
((org.postgresql.PGConnection)conn).addDataType("box3d",Class.forName("org.postgis.PGbox3d"));
```

Figure 2.5: Registering data type to the PostgreSQL connection [10]

And retrieving the Geometry and cast it to the correct type:

```
/*
* Retrieve the geometry as an object then cast it to the geometry type.
* Print things out.
*/
PGgeometry geom = (PGgeometry)r.getObject(1);
```

Figure 2.6: Retrieving Geometry from PostgreSQL database [10]

NOTE: The complete example can be seen in the Annex A.

2.5 Javadocs

Java languages counts with a documentation generator called *Javadocs*. It generates the HTML documentation based on the comments added to the code, that is a standard for Java[25][26].

The idea is to have the comments in the code with the information related to what we are commenting. We can see an example of a documented method:

```
/**
 * Gets the temporal type based on the string received.
 * @param value The string to be analyzed
 * @param temporalDataTypeName The name of the calling class
 * @return TemporalType
 * @throws SQLException
 */
public static TemporalType getTemporalType(String value,
   String temporalDataTypeName) throws SQLException {
....
}
```

And in the figure A.1, we can see the result after generating the javadocs.

getTemporalType

```
public static TemporalType getTemporalType(java.lang.String value, java.lang.String temporalDataTypeName) throws java.sql.SQLException
```

Gets the temporal type based on the string received.

Parameters:
`value` - The string to be analyzed
`temporalDataTypeName` - The name of the calling class

Returns:
TemporalType

Throws:
`java.sql.SQLException`

Figure 2.7: Generated documentation with Javadoc tool

2.6 Unit Tests and Integration Tests

Unit Tests are in charge of test the functionality of single modules of a software, verifying you are getting the expected result[30][38].

On the other hand *Integration Tests* check that different modules are working as expected together and identifying errors in any of the components[30][19].
In the table 2.8 we can see the differences between *Unit Testing* and *Integration Testing*.

Unit Testing	Integration Testing
Tests the single component of the whole system i.e. tests a unit in isolation.	Tests the system components working together i.e. test the collaboration of multiple units.
Faster to execute	Can run slow
No external dependency. Any external dependency is mocked or stubbed out.	Requires interaction with external dependencies (e.g. Database, hardware, etc.)
Simple	Complex
Conducted by developer	Conducted by tester
It is a type of white box testing	It is a type of black box testing
Carried out at the initial phase of testing and then can be performed anytime	Must be carried out after unit testing and before system testing
Cheap maintenance	Expensive maintenance
Begins from the module specification	Begins from the interface specification
Unit testing has a narrow scope as it just checks if each small piece of code is doing what it is intended to do.	It has a wider scope as it covers the whole application
The outcome of unit testing is detailed visibility of the code	The outcome of integration testing is the detailed visibility of the integration structure
Uncover the issues within the functionality of individual modules only. Does not exposes integration errors or system-wide issues.	Uncover the bugs arise when different modules interact with each other to form the overall system

Figure 2.8: Comparison between Unit Testing and Integration Testing [38]

2.7 JUnit

JUnit is an open-source Java testing framework, that gives a set of concepts and tools in basic and advanced level [37].

Some of the more important features from JUnit are:

- A long list of *Annotations* used for execution and identification of test methods.
- A variety of *Assertions* to check the expected results upon the current results.
- The *Test Runner* in charge of tests' execution [4].
- Basic template[4] to help with a basic structure.

- The *Test Suites* to group tests by testing functionality or order of execution.
- Reports with detailed information.

Junit5 has 3 main sub projects: JUnit Platform, JUnit Jupiter and JUnit Vintage [37] that contain many different modules.

1. JUnit Platform, in charge of TestEngine API, Console Launcher and JUnit Platform Suite Engine.
2. JUnit Jupiter, provides models for tests development and extensions.
3. JUnit Vintage, gives the possibility of running past versions 3 and 4.

2.8 SonarQube

SonarQube is an open-source platform for checking code quality and maintainability, by inspecting the code changes continuously, looking at bugs, vulnerabilities, code smells, duplication, testing coverage and use of standards [34]. The reports provide in dept recommendations and analysis, with graphs that show the evolution in time of the developed code. It can be integrated with tools as Gradle, Ant, Maven,etc.

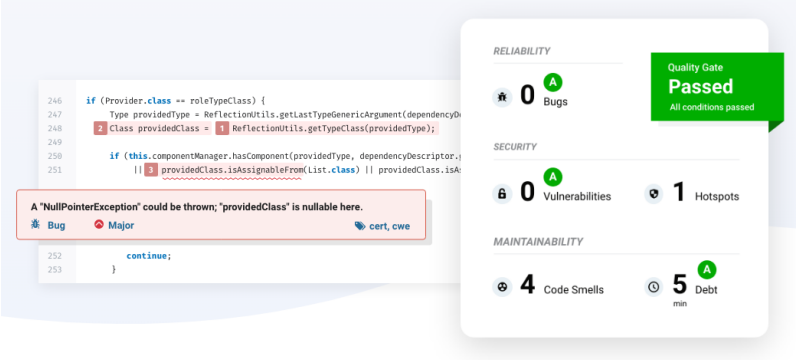

Figure 2.9: SonarQube functionality example[34]

Chapter 3

Design

The main requirements to add MobilityDB data types to PostgreSQL JDBC is that the classes representing these types must extend PGobject class of the package *org.postgresql.util* and also they need to be registered each time a connection is made, by calling *addDataType* method of the package *org.postgresql.PGConnection*.

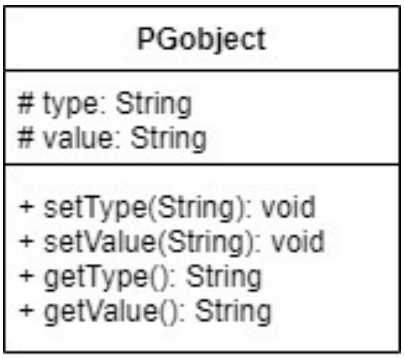

Figure 3.1: PGObject class [31]

The PGObject class (depicted in the figure 3.1)most important points are:

- By default Postgresql needs a constructor, that does not receive any parameters, so it is possible to create an empty object of any type, and then with the method *setValue* set the corresponding value to that object.
- The method *getValue* is called when the type is used as a parameter for a statement.
- The method *setValue* is called when reading results from a statement.

To fulfill the requirements, it was decided to create a base abstract class for MobilityDB data types called *DataType* that extends from PGObject and overrides the methods *getValue* and *setValue* making them abstract, so the child classes are forced to implement them.

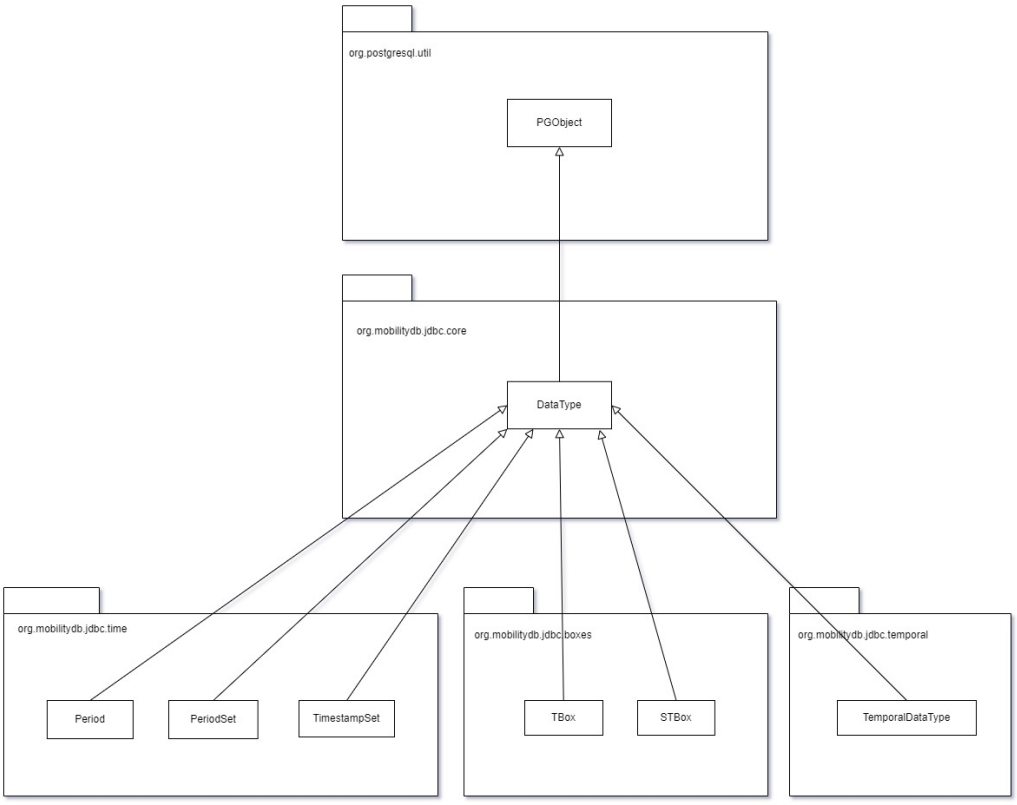

Figure 3.2: MobilityDB JDBC general diagram

Furthermore, this class is used by the singleton *DataTypeHandler* that helps to register all the types to the connection so it will not be required to register each one.

```
...
    con = DriverManager.getConnection(url, user, password);
    PGConnection pgConnection = (PGConnection) con;
    DataTypeHandler.INSTANCE.registerTypes(pgConnection);
...
```

Every MobilityDB type extends from *DataType*, and they needs to use the custom annotation *TypeName* for the registration to the PGConnection.

```
@TypeName(name = "period")
public class Period extends DataType {
...
}
```

The class *Datatype* is part of the package *org.mobilitydb.jdbc.core*, that also contains the helper class *DateTimeFormatHelper* that contains methods for handling datetime values.

The individual type implementation will be described in detail in the following chapter 4.

3.1 Code Validation

To validate that the solution was working correctly while implementing the data types, it was decided to use **SonarQube**, assuring good quality of the code and avoid common errors or introduction of bugs.

The current state of the MobilityDB-JDBC driver code is shown in the figure 3.3.

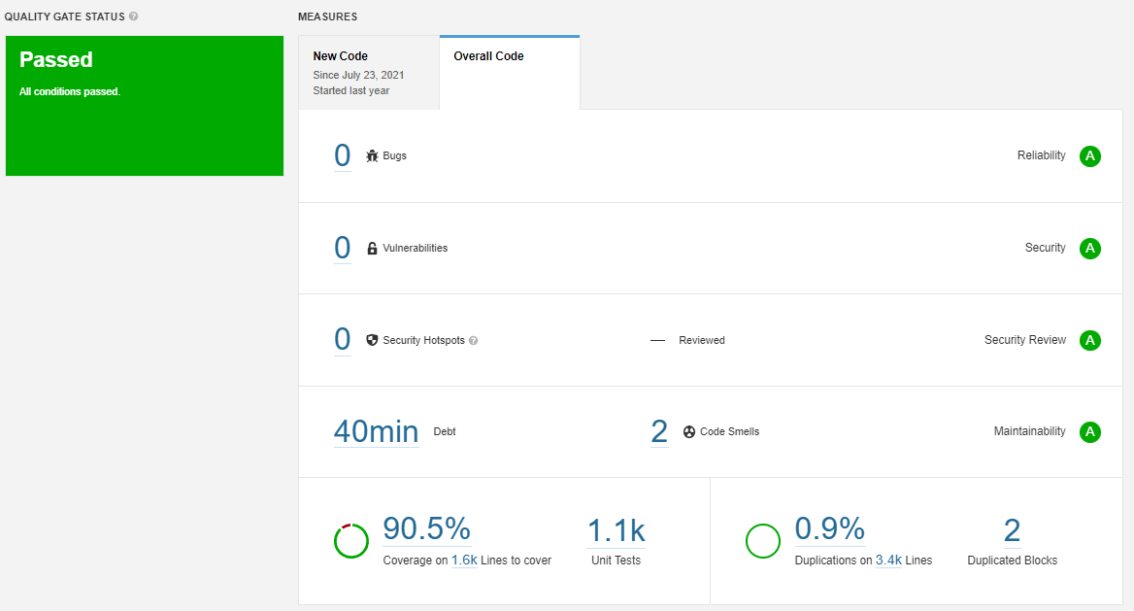

Figure 3.3: SonarQube analysis for MobilityDB-JDBC

In summary there are no bugs, vulnerabilities or security hotspots. There are 2 code smells related to the 2 duplicated blocks of code that means 0.9% of the whole project. Talking about the code coverage the 90.5% of the code is covered by more than 1100 unit tests.

3.1.1 Unit Tests

To test that each class and method works as expected and following the recommendation of **SonarQube** of having at least 80% of code coverage. The unit tests were used to prove that:

- The validations for data type are working fine.
- The constructors are building the data type objects correctly.
- The exceptions are thrown when they are expected.
- The methods are returning the expected values.

At the moment there are 1127 unit tests, the following code snippet is an example of an *unit test* of checking that a method is returning the correct value:

```
@Test
void testValueAtTimestamp() throws SQLException {
    ZoneOffset tz = ZoneOffset.of("+02:00");
    OffsetDateTime timestamp = OffsetDateTime.of(2001,1, 1, 8, 0,
       0, 0, tz);
    TIntSeq tIntSeq = new TIntSeq(
            "[89@2001-01-01 08:00:00+02, 3@2001-01-03 08:00:00+02,
               3@2001-01-04 08:00:00+02)");
    assertEquals(89, tIntSeq.valueAtTimestamp(timestamp));
}
```

The next example shows that an exception is expected when we sent invalid parameters to the constructor.

```
@Test
void testInvalidGeodetic() {
    Throwable exceptionThrown = assertThrows(SQLException.class,
       () -> {
        new STBox(new Point(1.0, 2.0), new Point(3.0, 4.0),true);
    });
    assertTrue(exceptionThrown.getMessage().contains("Geodetic
       coordinates need z value."));
}
```

3.1.2 Integration Tests

The integration test are used to verify that the MobilityDB-JDBC driver data types works correctly when reading and writing to a real database with MobilityDB extension. To accomplish this, we have the class *BaseIntegrationTest* that helps to:

- Create tables for each type, where we can save new records and do the cleanup of the data before each test.
- Create the connection to the database and register the MobilityDB-JDBC types.

At the moment there are 145 integration tests. Below we can see an example of an *integration test*, that is inserting to the database and then is reading from the database to check it is returning the correct data type:

```
@Test
void testStringsConstructor() throws Exception {
    String[] values = new String[] {"23.3@2001-01-01 08:20:00+03",
       "48.6@2001-01-03 18:08:00+03", "76.6@2001-01-03
       20:20:00+03"};

    TFloatInstSet tFloatInstSet = new TFloatInstSet(values);
    TFloat tFloat = new TFloat(tFloatInstSet);
```

```
    PreparedStatement insertStatement = con.prepareStatement(
            "INSERT INTO tbl_tfloat (temporaltype) VALUES (?);");
    insertStatement.setObject(1, tFloat);
    insertStatement.execute();
    insertStatement.close();

    PreparedStatement readStatement = con.prepareStatement(
            "SELECT temporaltype FROM tbl_tfloat WHERE
               temporaltype=?;");
    readStatement.setObject(1, tFloat);
    ResultSet rs = readStatement.executeQuery();

    if (rs.next()) {
        TFloat retrievedTFloat = (TFloat) rs.getObject(1);
        assertEquals(tFloat.getTemporal(),
           retrievedTFloat.getTemporal());
    } else {
        fail("TFloat was not retrieved.");
    }
    readStatement.close();
}
```

3.2 Example

In addition to the integration tests,with which we can use the MobilityDB-JDBC driver with a real database, it was required to create an example to use the MobilityDB-JDBC project as a dependency. Hence we have two different ways to test the driver:

1. The **Playground main class**, there it is possible to execute different SELECT queries and check that the correct type is retrieved.
2. The **Workshop main class**, uses MobilityDB-JDBC driver to follow some of the steps of the MobilityDB-Workshop[1][6], to work with a set of "real" data instead of dummy data, simulating a true environment. More details can be found in chapter 6.

[1]https://docs.mobilitydb.com/MobilityDB-workshop/master/mobilitydb-workshop.pdf

Chapter 4

Implementation

The MobilityDB-JDBC driver source code could be found in the Github repository **https://github.com/tatimilpo/MobilityDB-JDBC**.

For the implementation of the project there were many details that needed to be taken into account for the functionality and to keep the types standardized and follow the proposed design. In the following sections, we will describe the particular details of each class thay may include:

- Fields
- Constants
- Constructors
- Properties
- Overridden methods inherited from base classes
- Methods owned by the class

4.0.1 Shared Concepts

Inclusive and Exclusive Bounds

The bounds could be either *inclusive* "[ ]", where the first and/or last instant is part of the type and *exclusive* "( )" , where the first and/or last instant is not part of the type.

SRID

The spatial reference identifier SRID is used in "spatially enabled database as unique identifier for spatial data in a coordinate system"[36][41]. In case the SRID is not sent as parameter it will take the value 0(unknown) or 4326(WGS84[1])[43].

NOTE: The definition for the methods could be found in the Annex C and a summary of the methods implemented by each datatype in the Annex B

[1] World Geodetic System [42]

4.1 Time Types

Time types (package *org.mobilitydb.jdbc.time*) contain: **TimestampSet, Period** and **PeriodSet**. The three classes inherit from *DataType* and override the methods *getValue()* and *setValue()*. There is an aggregation relation between *Period* and *PeriodSet* classes, because the latter contains a list of *Periods*.
The figure 4.1 depicts the relations for the **Time Types**.

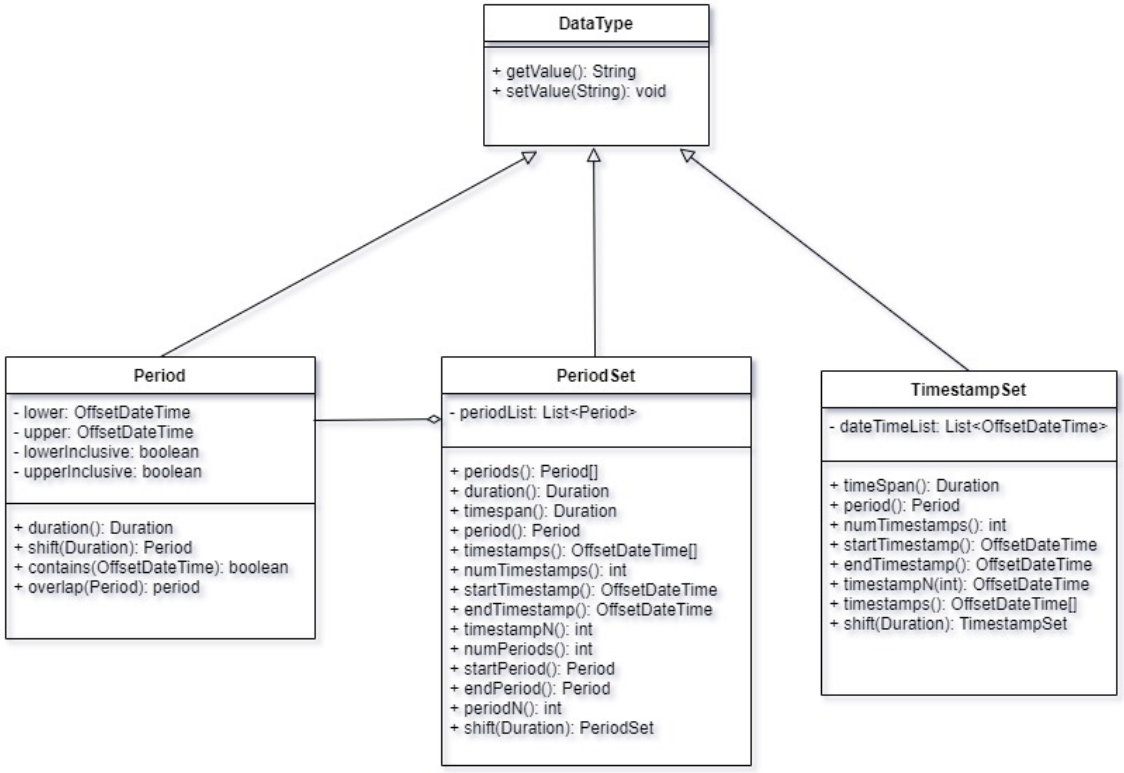

Figure 4.1: Time type architecture diagram

4.1.1 TimestampSet

A *TimestampSet* is a list of timestamps, in which all of the values should be different[45].

Among the validations we have:

- There should be at least one item in the *TimestampSet*.
- The timestamps should be in creasing order.
- All the timestamps should have a value.

4.1.1.1 Constructors

There are 3 different ways to create an object of the type *TimestampSet*:

1. The default constructor without parameters 3

```
TimestampSet timestampSet() = new TimestampSet();
```

2. The string constructor, that receives as single argument **a string** with a valid *TimestampSet* value.

```
\\ TimestampSet(String value)
TimestampSet timestampSet = new TimestampSet("{2019-09-08
   00:00:00+01, 2019-09-10 00:00:00+01, 2019-09-11
   00:00:00+01}");
```

3. The array of OffsetDateTime constructor, that receives *an array of OffsetDateTime or OffsetDateTime separated by a comma.*

```
\\ TimestampSet(OffsetDateTime... dateTimes)
TimestampSet timestampSet = new TimestampSet(
                OffsetDateTime.now(),
                OffsetDateTime.now().plusDays(1),
                OffsetDateTime.now().plusDays(2)
        );
```

4.1.1.2 Methods

The methods implemented for *TimestampSet* are:

- timeSpan()
- period()
- numTimestamps()
- startTimestamp()
- endTimestamp()
- timestampN(int index)
- timestamps()
- shift(Duration duration)

4.1.2 Period

A *Period* is a set of two timestamps with lower and upper bounds, where the *l*ower bound is 'true' and upper bound is 'false', if they are not explicitly sent as arguments[45].
Among the validations we have:

- The lower and upper value must be defined.
- The lower bound must be smaller or equal to the upper bound.
- The lower and upper bounds must be inclusive for an instant period.

4.1.2.1 Constructors

There are 4 different ways to create an object of the type *Period*:

1. The default constructor without parameters 3

```
Period period = new Period();
```

2. The string constructor, that receives as single argument **a string** with a valid *Period* value.

```
// Period(String value)
Period period = new Period("[2021-04-08 05:04:45+01,
    2021-09-10 10:00:00+01]");
```

3. The timestamps constructor, that receives **two OffsetDateTime values**, the lower timestamp (*lower*) followed by the upper timestamp (*upper*).

```
// Period(OffsetDateTime lower, OffsetDateTime upper)
Period period = new Period(OffsetDateTime.now(),
    OffsetDateTime.now().plusDays(2));
```

4. The timestamps and bounds constructor, that receives **an OffsetDateTime value, an OffsetDateTime value, a boolean, and a boolean**; the lower timestamp (*lower*), the upper timestamp (*upper*), the lower bound (*lowerInclusive*), and the upper bound (*upperInclusive*).

```
// Period(OffsetDateTime lower, OffsetDateTime upper, boolean
    lowerInclusive, boolean upperInclusive)
Period period = new Period(OffsetDateTime.now(),
    OffsetDateTime.now().plusDays(2), true, false);
```

4.1.2.2 Properties

The properties in *Period* are:

- **getLower**, the lower value.
- **getUpper**, the upper value.
- **isLowerInclusive**, if the lower bound is inclusive or not.
- **isUpperInclusive**, if the upper bound is inclusive or not.

4.1.2.3 Methods

The methods implemented for *Period* are:

- duration()
- shift(Duration duration)
- contains(OffsetDateTime dateTime)
- overlap(Period period)

4.1.3 PeriodSet

A *PeriodSet* is a list of periods, in which all of the values should be different[45]. Among the validations we have:

- The *Periods* in a *PeriodSet* should be sent in increasing order.
- There should be at least one item in the *PeriodSet*.
- All the *Periods* should have a value.
- The *Periods* of a *PeriodSet* cannot overlap.

4.1.3.1 Constructors

There are 3 different ways to create an object of the type *PeriodSet*:

1. The default constructor without parameters 3

```
PeriodSet periodSet = new PeriodSet();
```

2. The string constructor, that receives as single argument **a string** with a valid *PeriodSet* value.

```
// PeriodSet(String value)
Period period = new Period("{[2019-09-08 00:00:00+01,
   2019-09-10 00:00:00+01], [2019-09-11 00:00:00+01,
   2019-09-12 00:00:00+01]}");
```

3. The array of Periods constructor, that receives *an array of Periods or Periods separated by a comma.*

```
\\ PeriodSet(Period... periods)
PeriodSet periodSet = new PeriodSet(
                new Period("[2019-09-08 00:00:00+01,
                   2019-09-10 00:00:00+01]"),
                new Period("[2019-09-11 00:00:00+01,
                   2019-09-12 00:00:00+01]")
        );
```

4.1.3.2 Methods

- duration()
- timespan()
- numTimestamps()
- startTimestamp()
- endTimestamp()
- timestampN(int index)
- timestamps()

- period()
- periods()
- numPeriods()
- startPeriod()
- endPeriod()
- periodN(int n)
- shift(Duration duration)

4.2 Box Types

Box types (package *org.mobilitydb.jdbc.boxes*) contain: **TBox** and **STBox**. Both classes inherit from *DataType* and override the methods *getValue()* and *setValue()*. In addition *STBox* class has a composition relation with the class Point (*org.mobilitydb.jdbc.boxes.Point*), where *STBox* has 2-dimensional or 3-dimensional *Points*.
The figure 4.2 depicts the relations for the **Box Types**.

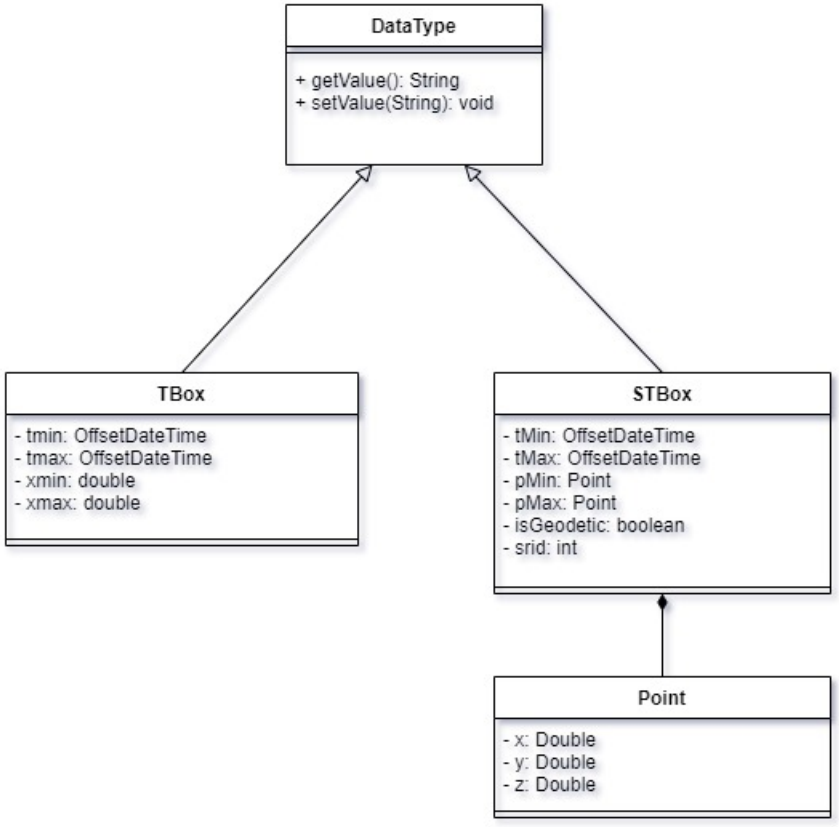

Figure 4.2: Box type architecture diagram

4.2.1 TBox

A *TBox* is a bounding box that could have a numeric dimension, a time dimension or both dimensions[45].
Among the validations we have:

- When x value is given, both xmin and xmax should have a value.
- When time is given, both tmin and tmax should have a value.

4.2.1.1 Constructors

There are 5 different ways to create an object of the type *TBox*:

1. The default constructor without parameters 3

```
TBox tBox = new TBox();
```

2. The string constructor, that receives as single argument **a string** with a valid *TBox* value.

```
// TBox(String value)
TBox tBox = new TBox("TBOX((50.0, 2021-07-08 06:04:32+02),
    (40.0, 2021-07-09 11:02:00+02))");
```

3. The only x dimension constructor, that receives **two doubles**, first the mininum x value (*xMin*) followed by the maximum x value (*xMax*).

```
// TBox(Double xMin, Double xMax)
TBox tBox = new TBox(3.0, 7.0);
```

4. The only time dimension constructor, that receives **two OffsetDateTime values**, the minimum time dimension (*tMin*) followed by the maximum time dimension (*tMax*).

```
// TBox(OffsetDateTime tMin, OffsetDateTime tMax)
OffsetDateTime tMin = OffsetDateTime.now().plusDays(-1);
OffsetDateTime tMax = OffsetDateTime.now();
TBox other = new TBox(tMin, tMax);
```

5. The value and time dimension constructor, that receives **a double, an OffsetDateTime value, a double and an OffsetDateTime value**; the mininum x value (*xMin*), the minimum time dimension (*tMin*), the maximum x value (*xMax*), and the maximum time dimension (*tMax*).

```
// TBox(Double xMin, OffsetDateTime tMin, Double xMax,
    OffsetDateTime tMax)
OffsetDateTime tMin = OffsetDateTime.now().plusDays(-1);
OffsetDateTime tMax = OffsetDateTime.now();
TBox tBox = new TBox(50.0, tMin, 40.0, tMax);
```

4.2.1.2 Properties

The properties in *TBox* are:

- **getXmin**, the minimum x value.
- **getXmax**, the maximum x value.
- **getTmin**, the minimum time.
- **getTmax**, the maximum time.

4.2.2 STBox

An *STBox* is a bounding box that could have a spatial value dimension, a time dimension or both dimensions[45][43]. The coordinates in the spatial value dimension could be 2-dimensional or 3 dimensional (x,y,z), that corresponds to *Point*. The *Point* class (package *com.mobilitydb.jdbc.boxes*) has 2 constructors for 2-dimensional (x,y) or 3-dimensional (x,y,z), and their properties. Moreover, the coordinates could be cartesian (flat coordinates)[2] or geodetic(angular coordinates)[3]. The SRID (definition in section 4.0.1) could be specified.
Among the validations we have:

- Both values in time dimension should have a value.
- Both x and y coordinates should have a value.
- Both z coordinates should have a value.
- Geodetic coordinates need z coordinate value.

4.2.2.1 Constructors

There are 14 different ways to create an object of the type *STBox*:

1. The default constructor without parameters 3

```
STBox stBox = new STBox();
```

2. The string constructor, that receives as single argument **a string** with a valid *STBox* value.

```
// STBox(String value)
STBox stbox = new STBox("STBOX T(, 2001-01-03 00:00:00+03), (,
    2001-01-03 00:00:00+03))");
```

[2]"Cartesian coordiante system, determine the position of a point with axis that intersect in an angle of 90 degrees, in 2-dimension or 3-dimensions." [14].
[3]"Geodetic coordinate system determine the position of a point with angular coordinates, that have a relation with spherical polar coordinates" [14]

3. The only value dimension constructor, that receives **two Points**, the first Point with the coordinates for the minimum bound *pMin* followed by another Point with the coordinates for the maximum bound *pMax*.

```
// STBox(Point pMin, Point pMax)
STBox stbox = new STBox(new Point(1.0,2.0,3.0), new
   Point(1.0,2.0,3.0));
```

4. The only time dimension constructor, that receives **two OffsetDateTime values**, the minimum time dimension (*tMin*) followed by the maximum time dimension (*tMax*).

```
// STBox(OffsetDateTime tMin, OffsetDateTime tMax)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(4);
STBox other = new STBox(tMin,tMax);
```

5. The value and time dimensions constructor, that receives **a Point, an OffsetDateTime value, a Point, and an OffsetDateTime value**, a Point with the coordinates for the minimum bound *pMin*, the minimum time dimension (*tMin*), a Point with the coordinates for the maximum bound *pMax* and the maximum time dimension (*tMax*).

```
// STBox(Point pMin, OffsetDateTime tMin, Point pMax,
   OffsetDateTime tMax)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(4);
STBox other = new STBox(new Point(1.0,2.0,3.0), tMin, new
   Point(1.0,2.0,3.0), tMax);
```

6. The only value dimension with geodetic coordinates constructor, that receives **two Points and a boolean**, the first Point with the coordinates for the minimum bound *pMin*, another Point with the coordinates for the maximum bound *pMax* and a boolean for geodetic coordinates (*isGeodetic*).

```
// STBox(Point pMin, Point pMax, boolean isGeodetic)
STBox stbox = new STBox(new Point(11.0,12.0,13.0), new
   Point(11.0,12.0,13.0),
                 true);
```

7. The only time dimension with geodetic coordinates constructor, that receives **two OffsetDateTime values and a boolean**, the minimum time dimension (*tMin*), the maximum time dimension (*tMax*) and a boolean for geodetic coordinates (*isGeodetic*).

```
// STBox(OffsetDateTime tMin, OffsetDateTime tMax, boolean
   isGeodetic)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(4);
STBox other = new STBox(tMin,tMax,false);
```

8. The value and time dimensions with geodetic coordinates constructor, that receives **a Point, an OffsetDateTime value, a Point, an OffsetDateTime value and a boolean**, a Point with the coordinates for the minimum bound *pMin*, the minimum time dimension (*tMin*), a Point with the coordinates for the maximum bound *pMax*, the maximum time dimension (*tMax*) and a boolean for geodetic coordinates (*isGeodetic*).

```
// STBox(STBox(Point pMin, OffsetDateTime tMin, Point pMax,
   OffsetDateTime tMax, boolean isGeodetic)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(1);
STBox other = new STBox(new Point(1.0,2.0,3.0), tMin, new
   Point(1.0,2.0,3.0), tMax, true);
```

9. The only value dimension with SRID constructor, that receives **two Points and an int**, the first Point with the coordinates for the minimum bound *pMin*, another Point with the coordinates for the maximum bound *pMax* and a int for the SRID (*srid*).

```
// STBox(Point pMin, Point pMax, int srid)
STBox stbox = new STBox(new Point(11.0,12.0), new
   Point(11.0,12.0), 4326);
                true);
```

10. The only time dimension with SRID constructor, that receives **two OffsetDateTime values and an int**, the minimum time dimension (*tMin*), the maximum time dimension (*tMax*) a int for the SRID (*srid*).

```
// STBox(OffsetDateTime tMin, OffsetDateTime tMax, int srid)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(1);
STBox other = new STBox(tMin,tMax,5676);
```

11. The value and time dimensions with SRID constructor, that receives **a Point, an OffsetDateTime value, a Point, an OffsetDateTime value and an int**, a Point with the coordinates for the minimum bound *pMin*, the minimum time dimension (*tMin*), a Point with the coordinates for the maximum bound *pMax*, the maximum time dimension (*tMax*) and an int for the SRID (*srid*).

```
// STBox(Point pMin, OffsetDateTime tMin, Point pMax,
   OffsetDateTime tMax, int srid)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(1);
STBox other = new STBox(new Point(1.0,2.0,3.0), tMin, new
   Point(1.0,2.0,3.0), tMax, 4326);
```

12. The only value dimension with geodetic coordinates and SRID constructor, that receives **two Points and an int**, the first Point with the coordinates for the minimum bound *pMin*, another Point with the coordinates for the maximum bound *pMax*, a boolean for geodetic coordinates (*isGeodetic*) and an int for the SRID (*srid*).

```
// STBox(Point pMin, Point pMax, boolean isGeodetic, int srid)
STBox stbox = new STBox(new Point(11.0,12.0), new
   Point(11.0,12.0), true, 4326);
```

13. The only time dimension with geodetic coordinates and SRID constructor, that receives **two OffsetDateTime values, a boolean and an int**, the minimum time dimension (*tMin*), the maximum time dimension (*tMax*), a boolean for geodetic coordinates (*isGeodetic*) and an int for the SRID (*srid*).

```
// STBox(OffsetDateTime tMin, OffsetDateTime tMax, boolean
   isGeodetic, int srid)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(4);
STBox other = new STBox(tMin,tMax,true, 4626);
```

14. The value and time dimensions with geodetic coordinates and SRID constructor, that receives **a Point, an OffsetDateTime value, a Point, an OffsetDateTime value, a boolean and an int**, a Point with the coordinates for the minimum bound *pMin*, the minimum time dimension (*tMin*), a Point with the coordinates for the maximum bound *pMax*, the maximum time dimension (*tMax*), a boolean for geodetic coordinates (*isGeodetic*) and an int for the SRID (*srid*).

```
// STBox(Point pMin, OffsetDateTime tMin, Point pMax,
   OffsetDateTime tMax, boolean isGeodetic, int srid)
OffsetDateTime tMin = OffsetDateTime.now();
OffsetDateTime tMax = OffsetDateTime.now().plusDays(1);
STBox other = new STBox(new Point(1.0,2.0,3.0), tMin, new
   Point(1.0,2.0,3.0), tMax, true,5676);
```

4.2.2.2 Properties

The properties in *TBox* are:

- **getXmin**, the minimum x value.
- **getXmax**, the maximum x value.
- **getYmin**, the minimum y value.
- **getYmax**, the maximum y value.
- **getZmin**, the minimum z value.
- **getZmax**, the maximum z value.
- **getTmin**, the minimum time.
- **getTmax**, the maximum time.
- **isGeodetic**, if the coordinates are geodetic.
- **getSrid**, the SRID value.

4.3 Temporal Types

TTemporal types contain: **TBool, TFloat, TGeogPoint, TGeomPoint, TInt** and **TText**. These data types required a special treatment because each of them could be of type *TInstant, TInstantSet, TSequence* or *TSequenceSet* but just on class could be registered to the PGConnection. Therefore it was required to create a different abstraction:

- The *TemporalType* ennumeration, will be used to determine the type of the Temporal class.
- The *Temporal* abstract class that, be extended by *TInstant, TInstantSet, TSequence* and *TSequenceSet*, and will define all the methods that need to be implemented.
- The *TemporalDataType* abstract class, inherits from DataType and overrides the methods *getValue()* and *setValue()* to use the *Temporal* class and will be extended by the temporal types.
- *TGeogPoint* and *TGeomPoint* inherit from TPoint class, that is used for managing the shared logic, handling SRID and the integration to PostGIS *Geometry and Point* classes.

The figure 4.3 depicts the relations for the **Temporal Types**.

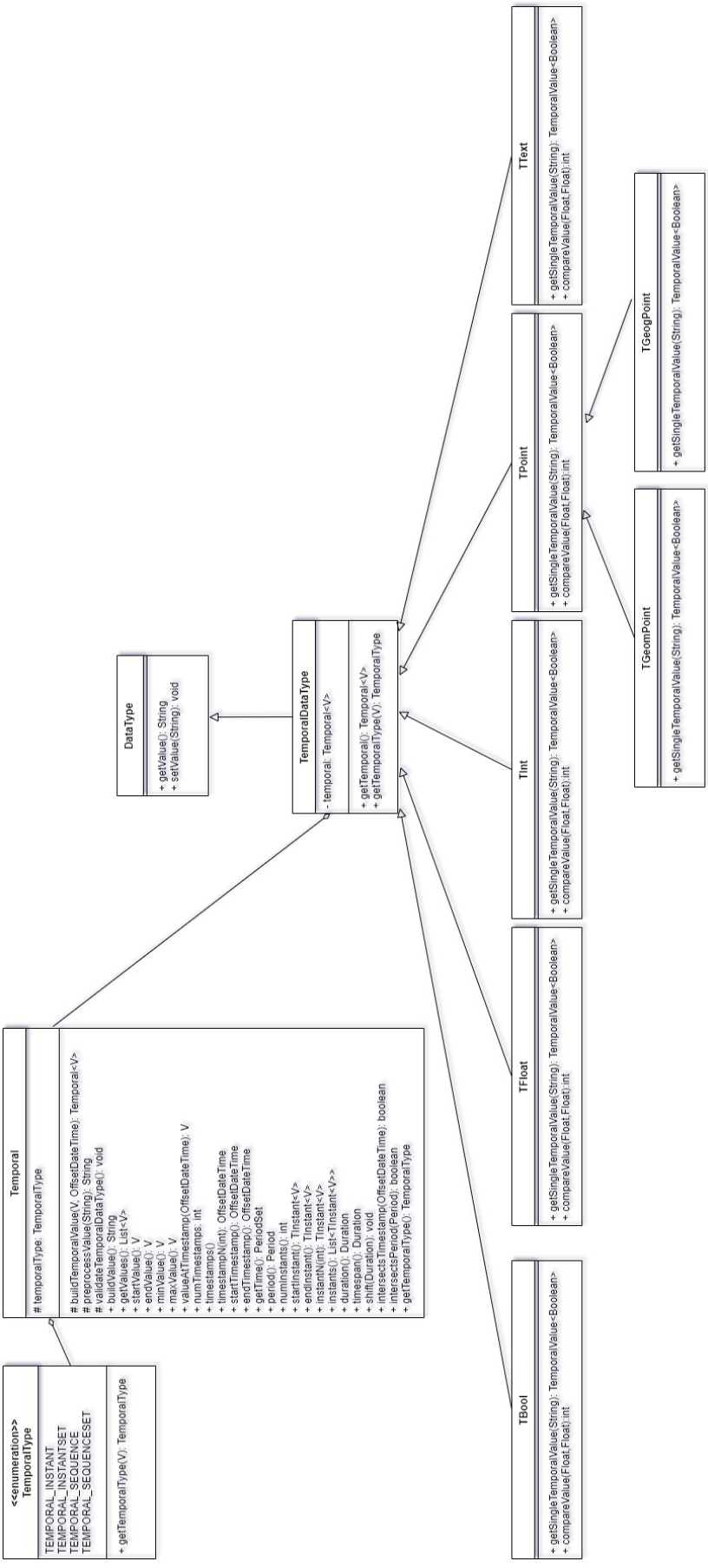

Figure 4.3: Temporal type architecture diagram

4.3.0.1 Shared concepts

Interpolations

The discrete base types TBool, TInt and TText will develop in a *stepwise* way, in opposition the types TFloat, TGeomPoint and TGeogPoint will develop either in a *stepwise* or in a *linear* way [43].

4.3.0.2 TBool

TBool is the data type registered in Postgresql connection, and it acts as a wrapper for the temporal types *TBoolInst, TBoolInstSet, TBoolSeq* and *TBoolSeqSet*.
For determining the temporal type, we need just to use the method *getTemporalType()*.

```
// Creating a TBoolInst
TBool tBool = new TBool("true@2019-09-08 06:04:32+02");
tBool.getTemporalType() //This will return TEMPORAL_INSTANT
```

In case we want to call any of the available methods inside the temporal type class we need to use the method *getTemporal()* and call the method directly, or it can be casted to the specific type.

```
tbool.getTemporal().getValue() //This will return true
((TBoolInst)tbool.getTemporal()).getValue() //This will also
   return true
```

4.3.0.2.1 Constructors

There are 3 constructors available to create an object of the type *TBool*:

1. The default constructor without parameters 3

   ```
   TBool tbool = new TBool();
   ```

2. The string constructor, that receives as single argument **a string** with a valid *TBool* value.

   ```
   // TBool(String value)
   TBool tBool = new TBool("{true@2001-01-01
      08:00:00+02,false@2001-01-03 08:00:00+02}");
   ```

3. The temporal constructor, that receives **a Temporal<Boolean>**, that could be of the type *TBoolInst, TBoolInstSet, TBoolSeq* and *TBoolSeqSet*.

   ```
   // TBool(Temporal<Boolean> temporal)
   TBool tBool = new TBool(new TBoolInstSet("{true@2001-01-01
      08:00:00+02, false@2001-01-03 08:00:00+02}"));
   ```

4.3.0.3 TFloat

TFloat is the data type registered in Postgresql connection, and it acts as a wrapper for the temporal types *TFloatInst, TFloatInstSet, TFloatSeq* and *TFloatSeqSet*.
For determining the temporal type, we need just to use the method *getTemporalType()*.

```
// Creating a TFloatInst
TFloat tFloat = new TFloat("10.5@2019-09-08 06:04:32+02");
tFloat.getTemporalType() //This will return TEMPORAL_INSTANT
```

In case we want to call any of the available methods inside the temporal type class we need to use the method *getTemporal()* and call the method directly, or it can be casted to the specific type.

```
tFloat.getTemporal().getValue() //This will return 10.5
((TFloatInst)tFloat.getTemporal()).getValue() //This will also
   return 10.5
```

4.3.0.3.1 Constructors

There are 3 constructors available to create an object of the type *TFloat*:

1. The default constructor without parameters 3

   ```
   TFloat tFloat = new TFloat();
   ```

2. The string constructor, that receives as single argument **a string** with a valid *TFloat* value.

   ```
   // TFloat(String value)
   TFloat tFloat = new TFloat("[16.98@2001-01-01 08:00:00+02,
      42.36@2001-01-03 08:00:00+02)");
   ```

3. The temporal constructor, that receives **a Temporal<Float>**, that could be of the type *TFloatInst, TFloatInstSet, TFloatSeq* and *TFloatSeqSet*.

   ```
   // TFloat(Temporal<Float> temporal)
   TFloat tFloat = new TFloat(new TFloatSeq("[16.98@2001-01-01
      08:00:00+02, 42.36@2001-01-03 08:00:00+02)"));
   ```

4.3.0.4 TInt

TInt is the data type registered in Postgresql connection, and it acts as a wrapper for the temporal types *TIntInst, TIntInstSet, TIntSeq* and *TIntSeqSet*.
For determining the temporal type, we need just to use the method *getTemporalType()*.

```
// Creating a TIntSeq
TInt tInt = new TInt("(1@2001-01-01 08:00:00+02, 2@2001-01-03
    08:00:00+02]");
tInt.getTemporalType() //This will return TEMPORAL_SEQUENCE
```

In case we want to call any of the available methods inside the temporal type class we need to use the method *getTemporal()* and call the method directly, or it can be casted to the specific type.

```
tInt.getTemporal().endValue() //This will return 2
((TIntSeq)tInt.getTemporal()).endValue() //This will also return 2
```

4.3.0.4.1 Constructors

There are 3 constructors available to create an object of the type *TInt*:

1. The default constructor without parameters 3

   ```
   TInt tInt = new TInt();
   ```

2. The string constructor, that receives as single argument **a string** with a valid *TInt* value.

   ```
   // TInt(String value)
   TInt tInt = new TInt("{[1@2001-01-01 08:00:00+02, 1@2001-01-03
       08:00:00+02)}");
   ```

3. The temporal constructor, that receives **a Temporal<Integer>**, that could be of the type *TIntInst, TIntInstSet, TIntSeq* and *TIntSeqSet*.

   ```
   // TInt(Temporal<Integer> temporal)
   TInt tInt = new TInt(new TIntInst("10@2019-09-08
       06:04:32+02"));
   ```

4.3.0.5 TGeomPoint

TGeomPoint is the data type registered in Postgresql connection, and it acts as a wrapper for the temporal types *TGeomPointInst, TGeomPointInstSet, TGeomPointSeq* and *TGeomPointSet*.
For determining the temporal type, we need just to use the method *getTemporalType()*.

```
// Creating a TGeomPointInst
TGeomPoint tGeomPoint = new TGeomPoint("Point(0 0)@2017-01-01
    08:00:05+02");
tGeomPoint.getTemporalType() //This will return TEMPORAL_INSTANT
```

In case we want to call any of the available methods inside the temporal type class we need to use the method *getTemporal()* and call the method directly, or it can be casted to the specific type.

```
tGeomPoint.getTemporal().numTimestamps() //This will return 1
((TGeomPointInst)tGeomPoint.getTemporal()).numTimestamps() //This
    will also return 1
```

4.3.0.5.1 Constructors

There are 3 constructors available to create an object of the type *TGeomPoint*:

1. The default constructor without parameters 3

   ```
   TGeomPoint tGeomPoint = new TGeomPoint();
   ```

2. The string constructor, that receives as single argument **a string** with a valid *TGeomPoint* value.

   ```
   // TGeomPoint(String value)
   TGeomPoint tGeomPoint = new TGeomPoint("Point(0 0)@2019-09-08
       06:04:32+02");
   ```

3. The temporal constructor, that receives **a Temporal<Point>**, that could be of the type *TGeomPointInst,TGeomPointInstSet, TGeomPointSeq* and *TGeomPointSeqSet*.

   ```
   // TGeomPoint(Temporal<Point> temporal)
   TGeomPoint tGeomPoint = new TGeomPoint(new
       TGeomPointInst("SRID=4326;Point(10.0 10.0)@2021-04-08
       05:04:45+01");
   ```

4.3.0.6 TGeogPoint

TGeogPoint is the data type registered in Postgresql connection, and it acts as a wrapper for the temporal types *TGeogPointInst, TGeogPointInstSet, TGeogPointSeq* and *TGeogPointSet*.
For determining the temporal type, we need just to use the method *getTemporalType()*.

```
// Creating a TGeogPointInst
TGeogPoint tGeogPoint = new TGeogPoint("[Point(0 0)@2001-01-01
    08:00:00+02, Point(1 1)@2001-01-03 08:00:00+02)");
tGeogPoint.getTemporalType() //This will return TEMPORAL_SEQUENCE
```

In case we want to call any of the available methods inside the temporal type class we need to use the method *getTemporal()* and call the method directly, or it can be casted to the specific type.

```
tGeogPoint.getTemporal().numSequences() //This will return 1
((TGeogPointInst)tGeogPoint.getTemporal()).numSequences() //This
    will also return 1
```

4.3.0.6.1 Constructors

There are 3 constructors available to create an object of the type *TGeogPoint*:

1. The default constructor without parameters 3

```
TGeogPoint tGeogPoint = new TGeogPoint();
```

2. The string constructor, that receives as single argument **a string** with a valid *TGeogPoint* value.

```
// TGeogPoint(String value)
TGeogPoint tGeogPoint = new TGeogPoint("SRID=4326;Point(10.0
    10.0)@2021-04-08 05:04:45+01");
```

3. The temporal constructor, that receives **a Temporal<Point>**, that could be of the type *TGeogPointInst,TGeogPointInstSet, TGeogPointSeq* and *TGeogPointSeqSet*.

```
// TGeogPoint(Temporal<Point> temporal)
TGeogPoint tGeogPoint = new TGeogPoint(new
    TGeogPointInst("SRID=4326;Point(10.0 10.0)@2021-04-08
    05:04:45+01");
```

4.3.0.7 TText

TText is the data type registered in Postgresql connection, and it acts as a wrapper for the temporal types *TTextInst, TTextInstSet, TTextSeq* and *TTextSeqSet*.
For determining the temporal type, we need just to use the method *getTemporalType()*.

```
// Creating a TTextSeqSet
TText tText = new TText("{(abcd@2001-01-01 08:00:00+02,
    efgh@2001-01-03 08:00:00+02], [ijkl@2001-01-04 08:00:00+02,
    mnop@2001-01-05 08:00:00+02, qrst@2001-01-06 08:00:00+02]}");
tText.getTemporalType() //This will return TEMPORAL_SEQUENCE_SET
```

In case we want to call any of the available methods inside the temporal type class we need to use the method *getTemporal()* and call the method directly, or it can be casted to the specific type.

```
tText.getTemporal().numSequences() //This will return 2
((TTextInst)tText.getTemporal()).numSequences() //This will also
    return 2
```

4.3.0.7.1 Constructors

There are 3 constructors available to create an object of the type *TBool*:

1. The default constructor without parameters 3

```
TText tText = new TText();
```

2. The string constructor, that receives as single argument **a string** with a valid *TText* value.

```
// TText(String value)
TText tText = new TText("{room@2001-01-01
   08:00:00+02,chair@2001-01-03 08:00:00+02}");
```

3. The temporal constructor, that receives **a Temporal<String>**, that could be of the type *TTextInst, TTextInstSet, TTextSeq* and *TTextSeqSet*.

```
// TText(Temporal<String> temporal)
TText tText = new TText(new TBoolInstSet("{Cloud@2001-01-01
   08:00:00+02, sky@2001-01-03 08:00:00+02}"));
```

4.3.1 Temporal Instant

Figure 4.4: TInstant architecture diagram

4.3.1.1 TBoolInst

There are 2 constructors available to create an object of the type *TBoolInst*:

1. The string constructor, that receives as single argument **a string** with a valid *TBoolInst* value.

```
// TBoolInst(String value)
TBoolInst tBoolInst = new TBoolInst("true@2019-09-08
    06:04:32+02");
```

2. The value and timestamp constructor, that receives **a boolean and an OffsetDateTime value**.

```
// TBoolInst(boolean value, OffsetDateTime time)
TBoolInst other = new TBoolInst(true, OffsetDateTime.now());
```

4.3.1.2 TFloatInst

There are 2 constructors available to create an object of the type *TFloatInst*:

1. The string constructor, that receives as single argument **a string** with a valid *TFloatInst* value.

```
// TFloatInst(String value)
TFloatInst tFloatInst = new TFloatInst("2.5@2019-09-08
    06:04:32+02");
```

2. The value and timestamp constructor, that receives **a double and an OffsetDateTime value**.

```
// TBoolInst(boolean value, OffsetDateTime time)
TFloatInst tFloatInst = new TFloatInst(8.5f,
    OffsetDateTime.now());
```

4.3.1.3 TIntInst

There are 2 constructors available to create an object of the type *TIntInst*:

1. The string constructor, that receives as single argument **a string** with a valid *TIntInst* value.

```
// TIntInst(String value)
TIntInst tIntInst = new TIntInst("9@2019-09-08 06:04:32+02");
```

2. The value and timestamp constructor, that receives **an int and an OffsetDateTime value**.

```
// TIntInst(int value, OffsetDateTime time)
TIntInst tIntInst = new TIntInst(88, OffsetDateTime.now());
```

4.3.1.4 TGeomPointInst

There are 2 constructors available to create an object of the type *TGeomPointInst*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeomPointInst* value.

```
// TGeomPointInst(String value)
TGeomPointInst tGeomPointInst = new TGeomPointInst("Point(0
    0)@2017-01-01 08:00:05+02");
```

2. The value and timestamp constructor, that receives **a Point and an OffsetDateTime value**.

```
// TGeomPointInst(Point value, OffsetDateTime time)
TGeomPointInst tGeomPointInst = new TGeomPointInst(new Point(0
    ,0), OffsetDateTime.now());
```

4.3.1.5 TGeogPointInst

There are 2 constructors available to create an object of the type *TGeogPointInst*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeogPointInst* value.

```
// TGeogPointInst(String value)
TGeogPointInst tGeogPointInst = new TGeogPointInst("Point(0
    0)@2017-01-01 08:00:05+02");
```

2. The value and timestamp constructor, that receives **a string and an OffsetDateTime value**.

```
// TTGeogPointInst(Point value, OffsetDateTime time)
TGeogPointInst tGeogPointInst = new TGeogPointInst(new Point(0
    ,0), OffsetDateTime.now());
```

4.3.1.6 TTextInst

There are 2 constructors available to create an object of the type *TTextInst*:

1. The string constructor, that receives as single argument **a string** with a valid *TTextInst* value.

```
// TTextInst(String value)
TTextInst tTextInst = new TTextInst("this is a text@2019-09-08
    06:04:32+02");
```

2. The value and timestamp constructor, that receives **a string and an OffsetDateTime value**.

```
// TTextInst(String value, OffsetDateTime time)
TTextInst tTextInst = new TTextInst("Another text",
    OffsetDateTime.now());
```

4.3.2 Temporal InstantSet

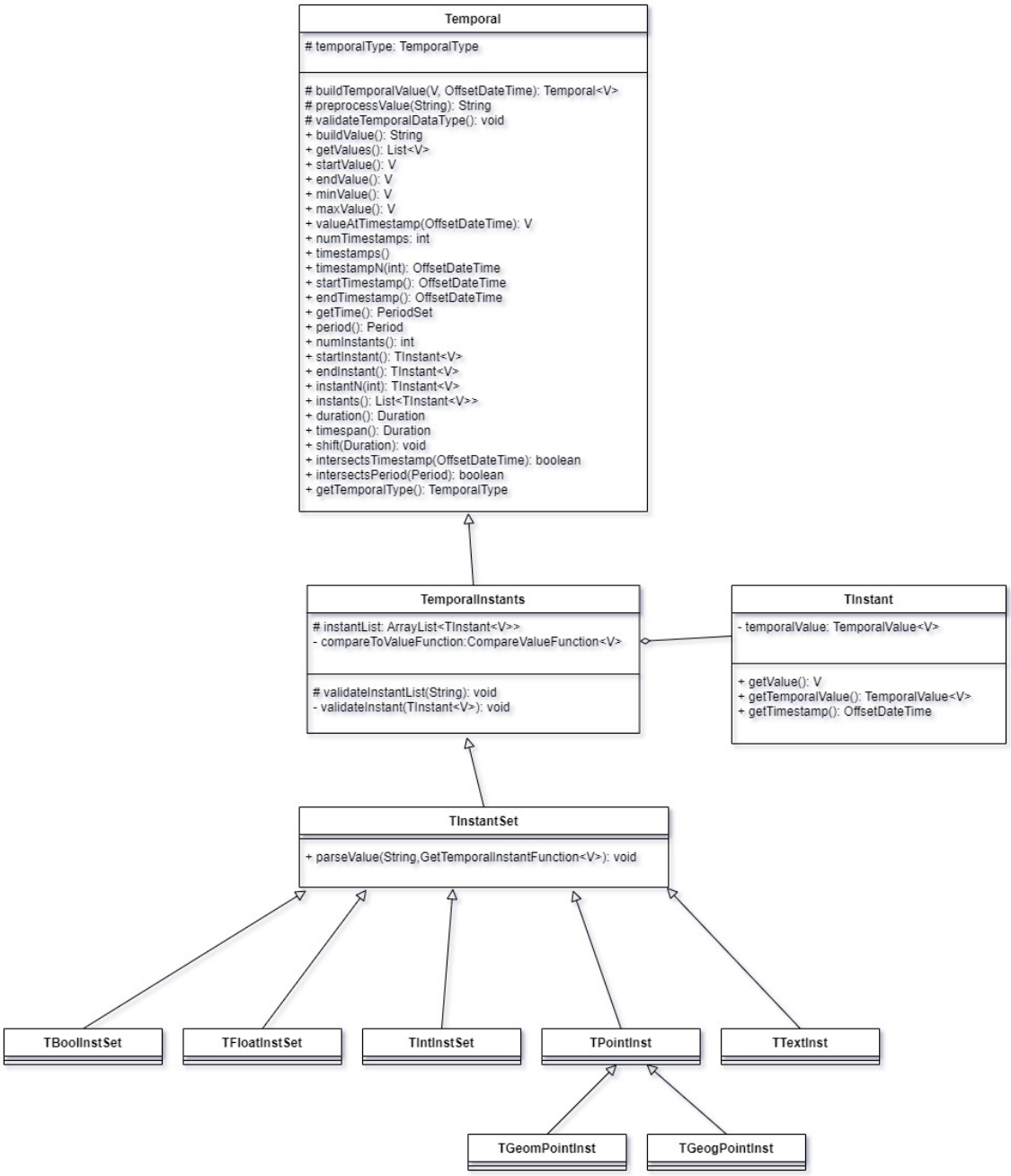

Figure 4.5: TInstantSet architecture diagram

4.3.2.1 TBoolInstSet

There are 3 constructors available to create an object of the type *TBoolInstSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TBoolInstSet* value.

```
// TBoolInstSet(String value)
```

```
TBoolInstSet tBoolInstSet = new
    TBoolInstSet("{false@2001-01-01 08:00:00+02,
    true@2001-01-03 08:00:00+02}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TBoolInstSet(String[] values)
String[] instants = new String[]{
            "true@2001-01-01 08:00:00+02",
            "false@2001-01-03 08:00:00+02"
        };
TBoolInstSet tBoolInstSet = new TBoolInstSet(instants);
```

3. The TBoolInst array constructor, that receives **an array of TBoolInst**.

```
// TBoolInstSet(TBoolInst[] values)
TBoolInst[] instants = new TBoolInst[]{
            new TBoolInst(false, OffsetDateTime.now()),
            new TBoolInst(true,
                OffsetDateTime.now().plusDays(2))
        };
TBoolInstSet tBoolInstSet = new TBoolInstSet(instants);
```

4.3.2.2 TFloatInstSet

There are 3 constructors available to create an object of the type *TFloatInstSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TFloatInstSet* value.

```
// TFloatInstSet(String value)
TFloatInstSet tFloatInstSet = new
    TFloatInstSet({1.8@2001-01-01 08:00:00+02, 2.8@2001-01-03
    08:00:00+02});
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TFloatInstSet(String[] values)
String[] instants = new String[]{
                "19.8@2001-01-01 08:00:00+02",
                "2.52@2001-01-03 08:00:00+02"
        };
TFloatInstSet tFloatInstSet = new TFloatInstSet(instants);
```

3. The TFloatInst array constructor, that receives **an array of TFloatInst**.

```
// TFloatInstSet(TFloatInst[] values)
TFloatInst[] instants = new TFloatInst[]{
```

```
                new TFloatInst(1.8f, OffsetDateTime.now()),
                new TFloatInst(2.8f,
                    OffsetDateTime.now().plusDays(2))
        };
TFloatInstSet tFloatInstSet = new TFloatInstSet(instants);
```

4.3.2.3 TIntInstSet

There are 3 constructors available to create an object of the type *TIntInstSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TIntInstSet* value.

```
// TIntInstSet(String value)
TIntInstSet tIntInstSet = new TIntInstSet("{1@2001-01-01
   08:00:00+02, 2@2001-01-03 08:00:00+02}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TIntInstSet(String[] values)
String[] instants = new String[]{
                "19.8@2001-01-01 08:00:00+02",
                "2.52@2001-01-03 08:00:00+02"
        };
TIntInstSet tIntInstSet = new TIntInstSet(instants);
```

3. The TIntInst array constructor, that receives **an array of TIntInst**.

```
// TIntInstSet(TIntInst[] values)
TIntInst[] instants = new TIntInst[]{
            new TIntInst(1, OffsetDateTime.now()),
            new TIntInst(2, OffsetDateTime.now().plusDays(2))
        };
TIntInstSet tIntInstSet = new TIntInstSet(instants);
```

4.3.2.4 TGeomPointInstSet

There are 5 constructors available to create an object of the type *TGeomPointInstSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeomPointInstSet* value.

```
// TGeomPointInstSet(String value)
TGeomPointInstSet tGeomPointInstSet = new
   TGeomPointInstSet("{Point(0 0)@2001-01-01 08:00:00+02,
   Point(1 1)@2001-01-03 08:00:00+02}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TGeomPointInstSet(String[] values)
String[] instants = new String[]{
            "Point(0 0)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointInstSet tGeomPointInstSet = new
   TGeomPointInstSet(instants);
```

3. The string array and SRID constructor, that receives **an int and an array of valid strings**.

```
// TGeomPointInstSet(int srid, String[] values)
String[] values = new String[]{
                "SRID=4326;Point(0 0)@2001-01-01 08:00:00+02",
                "SRID=4326;Point(1 1)@2001-01-03 08:00:00+02"
        };
TGeomPointInstSet tGeomPointInstSet = new
   TGeomPointInstSet(4326, instants);
```

4. The TGeomPointInst array constructor, that receives **an array of TGeomPointInst**.

```
// TGeomPointInstSet(TGeomPointInst[] values)
TGeomPointInst[] instants = new TGeomPointInst[]{
            new TGeomPointInst(new Point(0, 0),
               OffsetDateTime.now()),
            new TGeomPointInst(new Point (1, 1),
               OffsetDateTime.now().plusDays(2))
        };
TGeomPointInstSet tGeomPointInstSet = new
   TGeomPointInstSet(instants);
```

5. The TGeomPointInst array and SRID constructor, that receives **an int and an array of TGeomPointInst**.

```
// TGeomPointInstSet(int srid, TGeomPointInst[] values)
TGeomPointInst[] instants = new TGeomPointInst[]{
            new TGeomPointInst(new Point(0, 0),
               OffsetDateTime.now()),
            new TGeomPointInst(new Point (1, 1),
               OffsetDateTime.now().plusDays(2))
        };
TGeomPointInstSet tGeomPointInstSet = new
   TGeomPointInstSet(4326, instants);
```

4.3.2.5 TGeogPointInstSet

There are 5 constructors available to create an object of the type *TGeogPointInstSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeogPointInstSet* value.

```
// TGeogPointInstSet(String value)
TGeogPointInstSet tGeogPointInstSet = new
   TGeogPointInstSet("{Point(0 0)@2001-01-01 08:00:00+02,
   Point(1 1)@2001-01-03 08:00:00+02}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TGeogPointInstSet(String[] values)
String[] instants = new String[]{
            "Point(0 0)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointInstSet tGeogPointInstSet = new
   TGeogPointInstSet(instants);
```

3. The string array and SRID constructor, that receives **an int and an array of valid strings**.

```
// TGeogPointInstSet(int srid, String[] values)
String[] values = new String[]{
                "SRID=4326;Point(0 0)@2001-01-01 08:00:00+02",
                "SRID=4326;Point(1 1)@2001-01-03 08:00:00+02"
        };
TGeogPointInstSet tGeogPointInstSet = new
   TGeogPointInstSet(4326, instants);
```

4. The TGeogPointInst array constructor, that receives **an array of TGeogPointInst**.

```
// TGeogPointInstSet(TGeogPointInst[] values)
TGeogPointInst[] instants = new TGeogPointInst[]{
            new TGeogPointInst(new Point(0, 0),
               OffsetDateTime.now()),
            new TGeogPointInst(new Point (1, 1),
               OffsetDateTime.now().plusDays(2))
        };
TGeogPointInstSet tGeogPointInstSet = new
   TGeogPointInstSet(instants);
```

5. The TGeogPointInst array and SRID constructor, that receives **an int and an array of TGeogPointInst**.

```
// TGeogPointInstSet(int srid, TGeogPointInst[] values)
TGeogPointInst[] instants = new TGeogPointInst[]{
            new TGeogPointInst(new Point(0, 0),
                OffsetDateTime.now()),
            new TGeogPointInst(new Point (1, 1),
                OffsetDateTime.now().plusDays(2))
        };
TGeogPointInstSet tGeogPointInstSet = new
   TGeogPointInstSet(4326, instants);
```

4.3.2.6 TTextInstSet

There are 3 constructors available to create an object of the type *TTextInstSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TTextInstSet* value.

```
// TTextInstSet(String value)
TTextInstSet tTextInstSet = new
   TTextInstSet("{Random@2001-01-01 08:00:00+02,
   round@2001-01-03 08:00:00+02}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TTextInstSet(String[] values)
        String[] instants = new String[]{
                "ABCD@2001-01-01 08:00:00+02",
                "JKLM@2001-01-03 08:00:00+02"
        };
TTextInstSet tTextInstSet = new TTextInstSet(instants);
```

3. The TIntInst array constructor, that receives **an array of TIntInst**.

```
// TTextInstSet(TTextInst[] values)
TTextInst[] instants = new TTextInst[]{
                new TTextInst("Random", OffsetDateTime.now()),
                new TTextInst("Random2",
                    OffsetDateTime.now().plusDays(2))
        };
TTextInstSet tTextInstSet = new TTextInstSet(instants);
```

4.3.3 Temporal Sequence

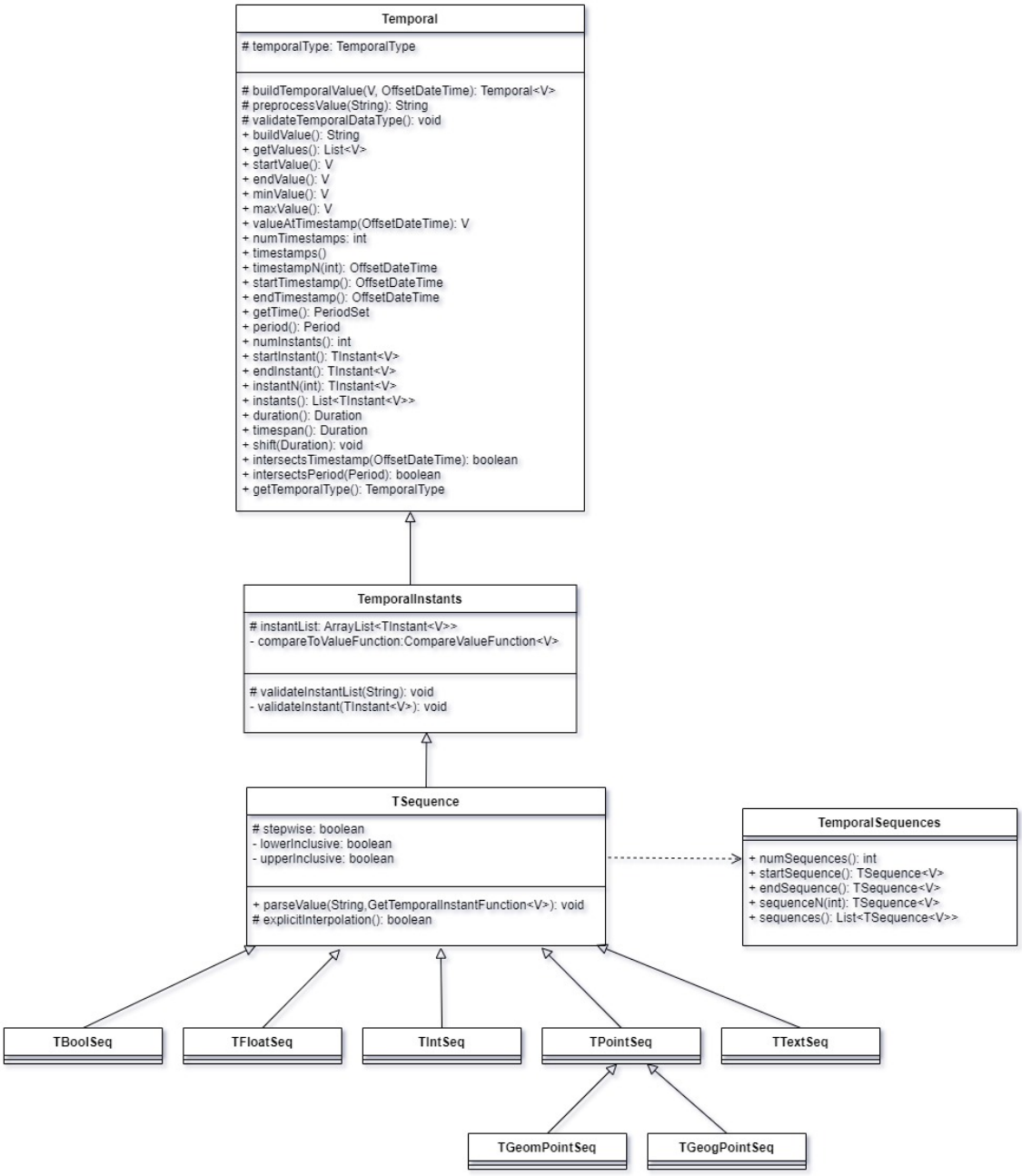

Figure 4.6: TSequence architecture diagram

4.3.3.1 TBoolSeq

There are 5 constructors available to create an object of the type *TBoolSeq*:

1. The string constructor, that receives as single argument **a string** with a valid *TBoolSeq* value.

```
// TBoolSeq(String value)
TBoolSeq tBoolSeq = new TBoolSeq("[true@2001-01-01
    08:00:00+02, true@2001-01-03 08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TBoolSeq(String[] values)
String[] instants = new String[]{
                "true@2001-01-01 08:00:00+02",
                "true@2001-01-03 08:00:00+02"
        };
TBoolSeq tBoolSeq = new TBoolSeq(instants);
```

3. The string array and bounds constructor, that receives **an array of valid strings and two booleans**. The first boolean corresponds to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TBoolSeq(String[] values, boolean lowerInclusive, boolean
   upperInclusive)
String[] instants = new String[]{
                "true@2001-01-01 08:00:00+02",
                "true@2001-01-03 08:00:00+02"
        };
TBoolSeq tBoolSeq = new TBoolSeq(instants, true, false);
```

4. The TBoolInst array constructor, that receives **an array of TBoolInst**.

```
// TBoolSeq(TBoolInst[] values)
TBoolInst[] instants = new TBoolInst[]{
            new TBoolInst(false, OffsetDateTime.now()),
            new TBoolInst(true,
                OffsetDateTime.now().plusDays(2))
        };
TBoolSeq tBoolSeq = new TBoolSeq(instants);
```

5. The TBoolInst array and bounds constructor, that receives **an array of TBoolInst and two booleans**. The first boolean corresponds to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TBoolSeq(TBoolInst[] values, boolean lowerInclusive,
   boolean upperInclusive)
TBoolInst[] instants = new TBoolInst[]{
            new TBoolInst(false, OffsetDateTime.now()),
            new TBoolInst(true,
                OffsetDateTime.now().plusDays(2))
        };
TBoolSeq tBoolSeq = new TBoolSeq(instants, true, false);
```

4.3.3.2 TFloatSeq

There are 9 constructors available to create an object of the type *TFloatSeq*:

1. The string constructor, that receives as single argument **a string** with a valid *TFloatSeq* value.

```
// TFloatSeq(String value)
TFloatSeq tFloatSeq = new TFloatSeq("[true@2001-01-01
   08:00:00+02, true@2001-01-03 08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TFloatSeq(String[] values)
String[] instants = new String[]{
                "2.5@2001-01-01 08:00:00+02",
                "2.8@2001-01-03 08:00:00+02"
        };
TFloatSeq tFloatSeq = new TFloatSeq(instants);
```

3. The string array and stepwise constructor, that receives **a boolean and an array of valid strings**.

```
// TFloatSeq(boolean isStepwise, String[] values)
String[] instants = new String[]{
                "2.5@2001-01-01 08:00:00+02",
                "2.8@2001-01-03 08:00:00+02"
        };
TFloatSeq tFloatSeq = new TFloatSeq(true, instants);
```

4. The string array and bounds constructor, that receives **a boolean, an array of valid strings and two booleans**. The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TFloatSeq(String[] values, boolean lowerInclusive, boolean
   upperInclusive)
String[] instants = new String[]{
                "2.5@2001-01-01 08:00:00+02",
                "2.8@2001-01-03 08:00:00+02"
        };
TFloatSeq tFloatSeq = new TFloatSeq(instants, true, false);
```

5. The string array, stepwise and bounds constructor, that receives **a boolean, an array of valid strings and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TFloatSeq(boolean isStepwise, String[] values, boolean
   lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
                "2.5@2001-01-01 08:00:00+02",
                "2.8@2001-01-03 08:00:00+02"
        };
TFloatSeq tFloatSeq = new TFloatSeq(true, instants, true,
   false);
```

6. The TFloatInst array constructor, that receives **an array of TFloatInst**.

```
// TFloatSeq(TFloatInst[] values)
TFloatInst[] instants = new TFloatInst[]{
                new TFloatInst(2.5f, OffsetDateTime.now()),
                new TFloatInst(2.8f,
                   OffsetDateTime.now().plusDays(2))
        };
TFloatSeq tFloatSeq = new TFloatSeq(instants);
```

7. The TFloatInst array and stepwise constructor, that receives **a boolean and an array of TFloatInst**.

```
// TFloatSeq(boolean isStepwise, TFloatInst[] values)
TFloatInst[] instants = new TFloatInst[]{
                new TFloatInst(2.5f, OffsetDateTime.now()),
                new TFloatInst(2.8f,
                   OffsetDateTime.now().plusDays(2))
        };
TFloatSeq tFloatSeq = new TFloatSeq(true, instants);
```

8. The TFloatInst array and bounds constructor, that receives **an array of TFloatInst and two booleans**.The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TFloatSeq(TFloatInst[] values, boolean lowerInclusive,
   boolean upperInclusive)
TFloatInst[] instants = new TFloatInst[]{
                new TFloatInst(2.5f, OffsetDateTime.now()),
                new TFloatInst(2.8f,
                   OffsetDateTime.now().plusDays(2))
        };
TFloatSeq tFloatSeq = new TFloatSeq(instants, true, false);
```

9. The TFloatInst array, stepwise and bounds constructor, that receives **a boolean, an array of TFloatInst and two booleans**. The first boolean corresponds

to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TFloatSeq(boolean isStepwise, TFloatInst[] values, boolean
   lowerInclusive, boolean upperInclusive)
TFloatInst[] instants = new TFloatInst[]{
                new TFloatInst(2.5f, OffsetDateTime.now()),
                new TFloatInst(2.8f,
                    OffsetDateTime.now().plusDays(2))
        };
TFloatSeq tFloatSeq = new TFloatSeq(true, instants, true,
   false);
```

4.3.3.3 TIntSeq

There are 5 constructors available to create an object of the type *TIntSeq*:

1. The string constructor, that receives as single argument **a string** with a valid *TIntSeq* value.

```
// TIntSeq(String value)
TIntSeq tIntSeq = new TIntSeq("[2@2001-01-01 08:00:00+02,
   2@2001-01-03 08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TIntSeq(String[] values)
String[] instants = new String[]{
                "2@2001-01-01 08:00:00+02",
                "2@2001-01-03 08:00:00+02"
        };
TIntSeq tIntSeq = new TIntSeq(instants);
```

3. The string array and bounds constructor, that receives **an array of valid strings and two booleans**. The first boolean corresponds to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TIntSeq(String[] values, boolean lowerInclusive, boolean
   upperInclusive)
String[] instants = new String[]{
                "2@2001-01-01 08:00:00+02",
                "2@2001-01-03 08:00:00+02"
        };
TIntSeq tIntSeq = new TIntSeq(instants, true, false);
```

4. The TBoolInst array constructor, that receives **an array of TBoolInst**.

```
// TIntSeq(TIntInst[] values)
TIntInst[] instants = new TIntInst[]{
                new TIntInst(2, OffsetDateTime.now()),
                new TIntInst(2,
                    OffsetDateTime.now().plusDays(2))
        };
TIntSeq tIntSeq = new TIntSeq(instants);
```

5. The TBoolInst array and bounds constructor, that receives **an array of TBoolInst and two booleans**. The first boolean corresponds to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TIntSeq(TIntInst[] values, boolean lowerInclusive, boolean
   upperInclusive)
TIntInst[] instants = new TIntInst[]{
                new TIntInst(2, OffsetDateTime.now()),
                new TIntInst(2,
                    OffsetDateTime.now().plusDays(2))
        };
TIntSeq tIntSeq = new TIntSeq(instants, true, false);
```

4.3.3.4 TGeomPointSeq

There are 17 constructors available to create an object of the type *TGeomPointSeq*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeomPointSeq* value.

```
// TGeomPointSeq(String value)
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq("[Point(1
   1)@2001-01-01 08:00:00+02, Point(2 2)@2001-01-03
   08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TGeomPointSeq(String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(true,
   instants);
```

3. The string array and stepwise constructor, that receives **a boolean and an array of valid strings**.

```
// TGeomPointSeq(boolean isStepwise, String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(true,
    instants);
```

4. The string array and SRID constructor, that receives **an int and an array of valid strings**.

```
// TGeomPointSeq(int srid, String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326,
    instants);
```

5. The string array, SRID and stepwise constructor, that receives **an int, a boolean and an array of valid strings**.

```
//TGeomPointSeq(int srid, boolean isStepwise, String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326, true,
    instants);
```

6. The string array and bounds constructor, that receives **an array of valid strings and two booleans**. The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TGeomPointSeq(String[] values, boolean lowerInclusive,
    boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(instants,
    true, false);
```

7. The string array, stepwise and bounds constructor, that receives **a boolean, an array of valid strings and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeomPointSeq(boolean isStepwise, String[] values, boolean
   lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(true,
   instants, true, false);
```

8. The string array, SRID and bounds constructor, that receives **an int, an array of valid strings and two booleans**. The booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeomPointSeq(int srid, String[] values, boolean
   lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326,
   instants, true, false);
```

9. The string array, SRID, stepwise and bounds constructor, that receives **an int, a boolean, an array of valid strings and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeomPointSeq(int srid, boolean isStepwise, String[]
   values, boolean lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326, true,
   instants, true, false);
```

10. The TGeomPointInst array constructor, that receives **an array of TGeomPointInst**.

```
// TGeomPointSeq(TGeomPointInst[] values)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
           OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
           OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(instants);
```

11. The TGeomPointInst array and stepwise constructor, that receives **a boolean and an array of TGeomPointInst**.

```
// TGeomPointSeq(boolean isStepwise, TGeomPointInst[] values)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(true,
    instants);
```

12. The TGeomPointInst array and SRID constructor, that receives **an int and an array of TGeomPointInst**.

```
// TGeomPointSeq(int srid, TGeomPointInst[] values)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326,
    instants);
```

13. The TGeomPointInst array, SRID and stepwise constructor, that receives **an int, a boolean and an array of TGeomPointInst**.

```
// TGeomPointSeq(int srid, boolean isStepwise,
    TGeomPointInst[] values)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326, true,
    instants);
```

14. The TGeomPointInst array and bounds constructor, that receives **an array of TGeomPointInst and two booleans**.The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TGeomPointSeq(TGeomPointInst[] values, boolean
    lowerInclusive, boolean upperInclusive)
TGeomPointInst[] instants = new TGeomPointInst[]{
```

```
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(instants,
    true, false);
```

15. The TGeomPointInst array, SRID and bounds constructor, that receives **an int, an array of TGeomPointInst and two booleans**.The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TGeomPointSeq(int srid, TGeomPointInst[] values, boolean
    lowerInclusive, boolean upperInclusive)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326,
    instants, true, false);
```

16. The TGeomPointInst array, stepwise and bounds constructor, that receives **a boolean, an array of TGeomPointInst and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeomPointSeq(boolean isStepwise, TGeomPointInst[] values,
    boolean lowerInclusive, boolean upperInclusive)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(true,
    instants, true, false);
```

17. The TGeomPointInst array, SRID, stepwise and bounds constructor, that receives **an int, a boolean, an array of TGeomPointInst and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeomPointSeq(int srid, boolean isStepwise,
   TGeomPointInst[] values, boolean lowerInclusive, boolean
   upperInclusive)
TGeomPointInst[] instants = new TGeomPointInst[]{
        new TGeomPointInst(new Point(1, 1),
           OffsetDateTime.now()),
        new TGeomPointInst(new Point(1, 1),
           OffsetDateTime.now().plusDays(2))
        };
TGeomPointSeq tGeomPointSeq = new TGeomPointSeq(4326, true,
   instants, true, false);
```

4.3.3.5 TGeogPointSeq

There are 17 constructors available to create an object of the type *TGeogPointSeq*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeogPointSeq* value.

```
// TGeogPointSeq(String value)
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq("[Point(1
   1)@2001-01-01 08:00:00+02, Point(2 2)@2001-01-03
   08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TGeogPointSeq(String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(true,
   instants);
```

3. The string array and stepwise constructor, that receives **a boolean and an array of valid strings**.

```
// TGeogPointSeq(boolean isStepwise, String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(true,
   instants);
```

4. The string array and SRID constructor, that receives **an int and an array of valid strings**.

```
// TGeogPointSeq(int srid, String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326,
    instants);
```

5. The string array, SRID and stepwise constructor, that receives **an int, a boolean and an array of valid strings**.

```
//TGeogPointSeq(int srid, boolean isStepwise, String[] values)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326, true,
    instants);
```

6. The string array and bounds constructor, that receives **an array of valid strings and two booleans**. The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TGeogPointSeq(String[] values, boolean lowerInclusive,
    boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(instants,
    true, false);
```

7. The string array, stepwise and bounds constructor, that receives **a boolean, an array of valid strings and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeogPointSeq(boolean isStepwise, String[] values, boolean
    lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(true,
    instants, true, false);
```

8. The string array, SRID and bounds constructor, that receives **an int, an array of valid strings and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeogPointSeq(int srid, String[] values, boolean
   lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326,
   instants, true, false);
```

9. The string array, SRID, stepwise and bounds constructor, that receives **an int, a boolean, an array of valid strings and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeogPointSeq(int srid, boolean isStepwise, String[]
   values, boolean lowerInclusive, boolean upperInclusive)
String[] instants = new String[]{
            "Point(1 1)@2001-01-01 08:00:00+02",
            "Point(2 2)@2001-01-03 08:00:00+02"
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326, true,
   instants, true, false);
```

10. The TGeogPointInst array constructor, that receives **an array of TGeogPointInst**.

```
// TGeogPointSeq(TGeogPointInst[] values)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(instants);
```

11. The TGeogPointInst array and stepwise constructor, that receives **a boolean and an array of TGeogPointInst**.

```
// TGeogPointSeq(boolean isStepwise, TGeogPointInst[] values)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now()),
```

```
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(true,
    instants);
```

12. The TGeogPointInst array and SRID constructor, that receives **an int and an array of TGeogPointInst**.

```
// TGeogPointSeq(int srid, TGeogPointInst[] values)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326,
    instants);
```

13. The TGeogPointInst array, SRID and stepwise constructor, that receives **an int, a boolean and an array of TGeogPointInst**.

```
// TGeogPointSeq(int srid, boolean isStepwise,
    TGeogPointInst[] values)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326, true,
    instants);
```

14. The TGeogPointInst array and bounds constructor, that receives **an array of TGeogPointInst and two booleans**.The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TGeogPointSeq(TGeogPointInst[] values, boolean
    lowerInclusive, boolean upperInclusive)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
            OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(instants,
    true, false);
```

15. The TGeogPointInst array, bounds and SRID constructor, that receives **an int, an array of TGeogPointInst and two booleans**.The first boolean to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TGeogPointSeq(int srid, TGeogPointInst[] values, boolean
   lowerInclusive, boolean upperInclusive)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
           OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
           OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326,
   instants, true, false);
```

16. The TGeogPointInst array, stepwise and bounds constructor, that receives **a boolean, an array of TGeogPointInst and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeogPointSeq(boolean isStepwise, TGeogPointInst[] values,
   boolean lowerInclusive, boolean upperInclusive)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
           OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
           OffsetDateTime.now().plusDays(2))
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(true,
   instants, true, false);
```

17. The TGeogPointInst array, SRID, stepwise and bounds constructor, that receives **an int, a boolean, an array of TGeogPointInst and two booleans**. The first boolean corresponds to stepwise *isStepwise*, the next booleans correspond to the lower bound *lowerInclusive* and to the upper bound *upperInclusive*.

```
// TGeogPointSeq(int srid, boolean isStepwise,
   TGeogPointInst[] values, boolean lowerInclusive, boolean
   upperInclusive)
TGeogPointInst[] instants = new TGeogPointInst[]{
        new TGeogPointInst(new Point(1, 1),
           OffsetDateTime.now()),
        new TGeogPointInst(new Point(1, 1),
           OffsetDateTime.now().plusDays(2))
```

```
        };
TGeogPointSeq tGeogPointSeq = new TGeogPointSeq(4326, true,
    instants, true, false);
```

4.3.3.6 TTextSeq

There are 5 constructors available to create an object of the type *TTextSeq*:

1. The string constructor, that receives as single argument **a string** with a valid *TTextSeq* value.

```
// TTextSeq(String value)
TTextSeq tTextSeq = new TTextSeq("[This is a text@2001-01-01
    08:00:00+02, This is a text@2001-01-03 08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TTextSeq(String[] values)
String[] instants = new String[]{
                "This is a text@2001-01-01 08:00:00+02",
                "This is other text@2001-01-03 08:00:00+02"
        };
TTextSeq tTextSeq = new TTextSeq(instants);
```

3. The string array and bounds constructor, that receives **an array of valid strings and two booleans**. The first boolean corresponds to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TTextSeq(String[] values, boolean lowerInclusive, boolean
    upperInclusive)
String[] instants = new String[]{
                "This is a text@2001-01-01 08:00:00+02",
                "This is other text@2001-01-03 08:00:00+02"
        };
TTextSeq tTextSeq = new TTextSeq(instants, true, false);
```

4. The TTextInst array constructor, that receives **an array of TTextInst**.

```
// TTextSeq(TTextInst[] values)
TTextInst[] instants = new TTextInst[]{
                new TTextInst("This is a text",
                    OffsetDateTime.now()),
                new TTextInst("This is a text",
                    OffsetDateTime.now().plusDays(2))
        };
TTextSeq tTextSeq = new TTextSeq(instants);
```

5. The TTextInst array and bounds constructor, that receives **an array of TTextInst and two booleans**. The first boolean corresponds to the lower bound *lowerInclusive* and the second boolean corresponds to the upper bound *upperInclusive*.

```
// TTextSeq(TTextInst[] values, boolean lowerInclusive,
   boolean upperInclusive)
TTextInst[] instants = new TTextInst[]{
                new TTextInst("This is a text",
                    OffsetDateTime.now()),
                new TTextInst("This is a text",
                    OffsetDateTime.now().plusDays(2))
        };
TTextSeq tTextSeq = new TTextSeq(instants, true, false);
```

4.3.4 Temporal SequenceSet

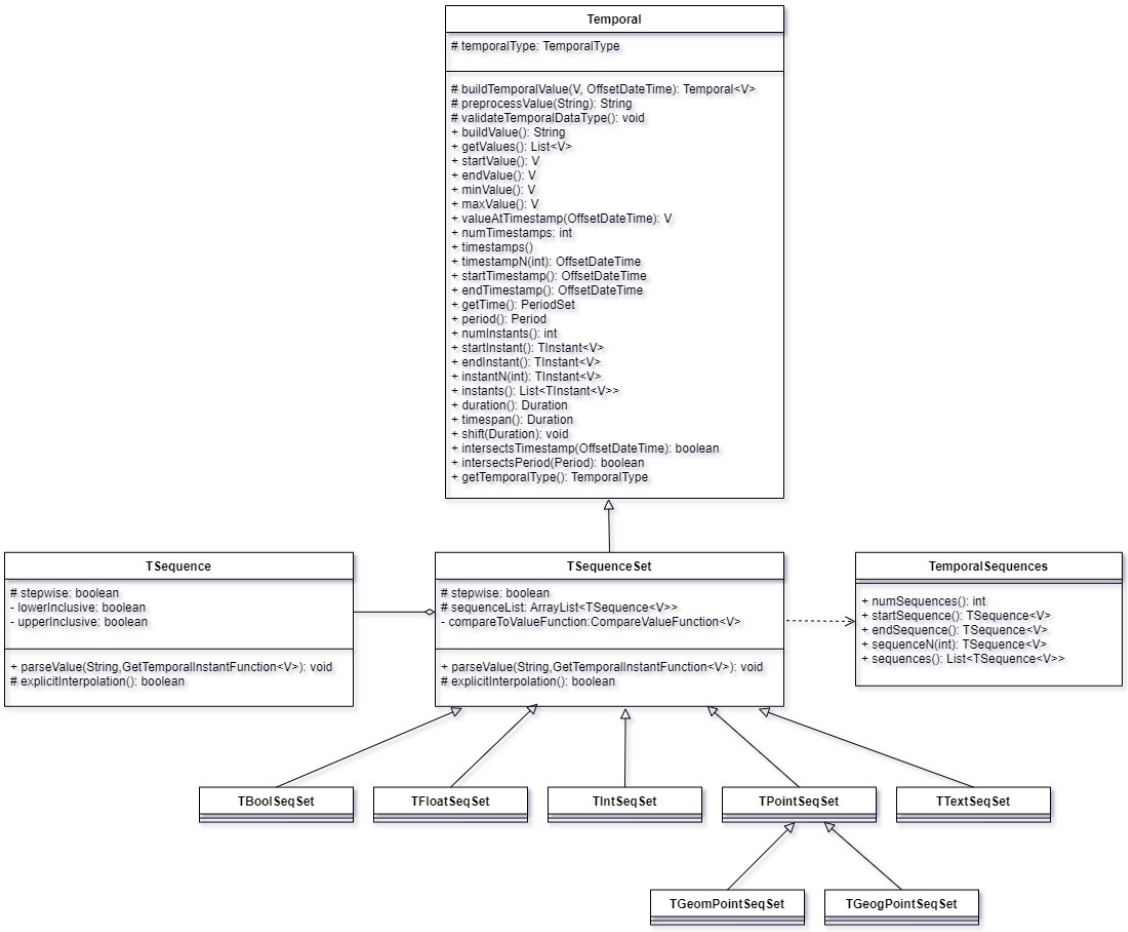

Figure 4.7: TSequenceSet architecture diagram

4.3.4.1 TBoolSeqSet

There are 3 constructors available to create an object of the type *TBoolSeqSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TBoolSeqSet* value.

```
// TBoolSeqSet(String value)
TBoolSeqSet tBoolSeqSet = new TBoolSeqSet("{(false@2001-01-01
   08:00:00+02, true@2001-01-03 08:00:00+02],
   [false@2001-01-04 08:00:00+02, true@2001-01-05
   08:00:00+02, true@2001-01-06 08:00:00+02]}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TBoolSeqSet(String[] values)
String[] sequences = new String[]{
    "(false@2001-01-01 08:00:00+02, true@2001-01-03
       08:00:00+02]",
    "[false@2001-01-04 08:00:00+02, true@2001-01-05
       08:00:00+02, true@2001-01-06 08:00:00+02]"
    };
TBoolSeqSet tBoolSeqSet = new TBoolSeqSet(sequences);
```

3. The TBoolSeq array constructor, that receives **an array of TBoolSeq**.

```
// TBoolSeqSet(TBoolSeq[] values)
TBoolSeq[] sequences = new TBoolSeq[]{
        new TBoolSeq("(false@2001-01-01 08:00:00+02,
           true@2001-01-03 08:00:00+02]"),
        new TBoolSeq("[false@2001-01-04 08:00:00+02,
           true@2001-01-05 08:00:00+02, true@2001-01-06
           08:00:00+02]")
        };
TBoolSeqSet tBoolSeqSet = new TBoolSeqSet(sequences);
```

4.3.4.2 TFloatSeqSet

There are 5 constructors available to create an object of the type *TFloatSeqSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TFloatSeqSet* value.

```
// TFloatSeqSet(String value)
TFloatSeqSet tFloatSeqSet = new TFloatSeqSet("{[1.8@2001-01-01
   08:00:00+02, 1.9@2001-01-03 08:00:00+02), [2.3@2001-01-04
   08:00:00+02, 3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
   08:00:00+02]}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TFloatSeqSet(String[] values)
String[] stringSequences = new String[]{
    "Interp=Stepwise;(1.8@2001-01-01 08:00:00+02,
       1.9@2001-01-03 08:00:00+02]",
    "Interp=Stepwise;[2.3@2001-01-04 08:00:00+02,
       3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
       08:00:00+02]"
    };
TFloatSeqSet tFloatSeqSet = new TFloatSeqSet(sequences);
```

3. The string array and stepwise constructor, that receives **a boolean and an array of valid strings**.

```
// TFloatSeqSet(boolean stepwise, TFloatSeq[] values)
String[] stringSequences = new String[]{
    "Interp=Stepwise;(1.8@2001-01-01 08:00:00+02,
       1.9@2001-01-03 08:00:00+02]",
    "Interp=Stepwise;[2.3@2001-01-04 08:00:00+02,
       3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
       08:00:00+02]"
    };
TFloatSeqSet tFloatSeqSet = new TFloatSeqSet(true, sequences);
```

4. The TFloatSeq array constructor, that receives **an array of TFloatSeq**.

```
// TFloatSeqSet(TFloatSeq[] values)
TFloatSeq[] sequences = new TFloatSeq[]{
        "[1.8@2001-01-01 08:00:00+02, 1.9@2001-01-03
           08:00:00+02)",
        "[2.3@2001-01-04 08:00:00+02, 3.4@2001-01-05
           08:00:00+02, 3.5@2001-01-06 08:00:00+02]"
        };
TFloatSeqSet tFloatSeqSet = new TFloatSeqSet(sequences);
```

5. The TFloatSeq array and stepwise constructor, that receives **a boolean and an array of TFloatSeq**.

```
// TFloatSeqSet(boolean stepwise, String[] values)
TFloatSeq[] sequences = new TFloatSeq[]{
        new TFloatSeq("Interp=Stepwise;(1.8@2001-01-01
           08:00:00+02, 1.9@2001-01-03 08:00:00+02]"),
        new TFloatSeq("Interp=Stepwise;[2.3@2001-01-04
           08:00:00+02, 3.4@2001-01-05
           08:00:00+02,3.5@2001-01-06 08:00:00+02]")
        };
TFloatSeqSet tFloatSeqSet = new TFloatSeqSet(true, sequences);
```

4.3.4.3 TIntSeqSet

There are 3 constructors available to create an object of the type *TIntSeqSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TIntSeqSet* value.

```
// TIntSeqSet(String value)
TIntSeqSet tIntSeqSet = new TIntSeqSet("{[1@2001-01-01
   08:00:00+02, 1@2001-01-03 08:00:00+02), [2@2001-01-04
   08:00:00+02, 3@2001-01-05 08:00:00+02, 3@2001-01-06
   08:00:00+02]}");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TIntSeqSet(String[] values)
String[] sequences = new String[]{
    "[1@2001-01-01 08:00:00+02, 1@2001-01-03 08:00:00+02)",
    "[2@2001-01-04 08:00:00+02, 3@2001-01-05 08:00:00+02,
       3@2001-01-06 08:00:00+02]"
    };
TIntSeqSet tIntSeqSet = new TIntSeqSet(sequences);
```

3. The TIntSeq array constructor, that receives **an array of TIntSeq**.

```
// TIntSeqSet(TIntSeq[] values)
TIntSeq[] sequences = new TIntSeq[]{
        new TIntSeq("[1@2001-01-01 08:00:00+02, 1@2001-01-03
           08:00:00+02)"),
        new TIntSeq("[2@2001-01-04 08:00:00+02, 3@2001-01-05
           08:00:00+02, 3@2001-01-06 08:00:00+02]")
        };
TIntSeqSet tIntSeqSet = new TIntSeqSet(sequences);
```

4.3.4.4 TGeomPointSeqSet

There are 9 constructors available to create an object of the type *TGeomPointSeqSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeomPointSeqSet* value.

```
// TGeomPointSeqSet(String value)
TGeomPointSeqSet tGeomPointSeqSet = new
   TGeomPointSeqSet("Interp=Stepwise;[POINT(1 1)@2001-01-01
   08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TGeomPointSeq(String[] values)
String[] sequences = new String[]{
        "[POINT(1 1)@2001-01-01 08:00:00+02, POINT(1
           1)@2001-01-03 08:00:00+02)", "[POINT(1
           1)@2001-01-04 08:00:00+02, POINT(2 2)@2001-01-05
           08:00:00+02,POINT(3 3)@2001-01-06 08:00:00+02]"
        };
TGeomPointSeqSet tGeomPointSeqSet = new
   TGeomPointSeqSet(sequences);
```

3. The string array and stepwise constructor, that receives **a boolean and an array of valid strings**.

```
// TGeomPointSeqSet(boolean stepwise, String[] values)
String[] sequences = new String[]{
    "Interp=Stepwise;(1.8@2001-01-01 08:00:00+02,
       1.9@2001-01-03 08:00:00+02]",
    "Interp=Stepwise;[2.3@2001-01-04 08:00:00+02,
       3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
       08:00:00+02]"
    };
TGeomPointSeqSet tGeomPointSeqSet = new TGeomPointSeqSet(true,
   sequences);
```

4. The string array and SRID constructor, that receives **an int and an array of valid strings**.

```
// TGeomPointSeqSet(int srid, String[] values)
 String[] sequences = new String[]{
    "SRID=4326;[POINT(1 1)@2001-01-01 08:00:00+02, POINT(1
       1)@2001-01-03 08:00:00+02)",
    "SRID=4326;[POINT(1 1)@2001-01-04 08:00:00+02, POINT(2
       2)@2001-01-05 08:00:00+02, POINT(3 3)@2001-01-06
       08:00:00+02]"
    };
TGeomPointSeqSet tGeomPointSeqSet = new TGeomPointSeqSet(4326,
   sequences);
```

5. The string array, SRID and stepwise constructor, that receives **an int, a boolean and an array of valid strings**.

```
// TGeomPointSeqSet(int srid, boolean stepwise, String[]
   values)
String[] sequences = new String[]{
    "Interp=Stepwise;(1.8@2001-01-01 08:00:00+02,
       1.9@2001-01-03 08:00:00+02]",
```

```
    "Interp=Stepwise;[2.3@2001-01-04 08:00:00+02,
       3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
       08:00:00+02]"
    };
TGeomPointSeqSet tGeomPointSeqSet = new TGeomPointSeqSet(4326,
   true, sequences);
```

6. The TGeomPointSeq array constructor, that receives **an array of TGeomPointSeq**.

```
// TGeomPointSeqSet(TGeomPointSeq[] values)
TGeomPointSeq[] sequences = new TGeomPointSeq[]{
    new TGeomPointSeq("[POINT(1 1)@2001-01-01 08:00:00+02,
       POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeomPointSeq("[POINT(1 1)@2001-01-04 08:00:00+02,
       POINT(2 2)@2001-01-05 08:00:00+02, POINT(3
       3)@2001-01-06 08:00:00+02])
    }";
TGeomPointSeqSet tGeomPointSeqSet = new
   TGeomPointSeqSet(sequences);
```

7. The TGeomPointSeq array and stepwise constructor, that receives **a boolean and an array of TGeomPointSeq**.

```
// TGeomPointSeqSet(boolean stepwise, TGeomPointSeq[] values)
TGeomPointSeq[] sequences = new TGeomPointSeq[]{
    new TGeomPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-01
       08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeomPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-04
       08:00:00+02, POINT(2 2)@2001-01-05 08:00:00+02,
       POINT(3 3)@2001-01-06 08:00:00+02]")
        };
TGeomPointSeqSet tGeomPointSeqSet = new
   TGeomPointSeqSett(true, sequences);
```

8. The TGeomPointSeq array and SRID constructor, that receives **a int and an array of TGeomPointSeq**.

```
// TGeomPointSeqSet(int srid, TGeomPointSeq[] values)
TGeomPointSeq[] sequences = new TGeomPointSeq[]{
    new TGeomPointSeq("SRID=4326;[POINT(1 1)@2001-01-01
       08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeomPointSeq("SRID=4326;[POINT(1 1)@2001-01-04
       08:00:00+02, POINT(2 2)@2001-01-05 08:00:00+02,
       POINT(3 3)@2001-01-06 08:00:00+02]")
        };
TGeomPointSeqSet tGeomPointSeqSet = new TGeomPointSeqSet(4326,
   sequences);
```

9. The TGeomPointSeq array, SRID and stepwise constructor, that receives **an int, a boolean and an array of TGeomPointSeq**.

```
// TGeomPointSeqSet(int srid, boolean stepwise,
   TGeomPointSeq[] values)
TGeomPointSeq[] sequences = new TGeomPointSeq[]{
    new TGeomPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-01
       08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeomPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-04
       08:00:00+02, POINT(2 2)@2001-01-05 08:00:00+02,
       POINT(3 3)@2001-01-06 08:00:00+02]")
        };
TGeomPointSeqSet tGeomPointSeqSet = new TGeomPointSeqSet(4326,
   true, sequences);
```

4.3.4.5 TGeogPointSeqSet

There are 9 constructors available to create an object of the type *TGeogPointSeqSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TGeogPointSeqSet* value.

```
// TGeogPointSeqSet(String value)
TGeogPointSeqSet tGeogPointSeqSet = new
   TGeogPointSeqSet("Interp=Stepwise;[POINT(1 1)@2001-01-01
   08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)");
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TGeogPointSeqSet(String[] values)
String[] sequences = new String[]{
        "[POINT(1 1)@2001-01-01 08:00:00+02, POINT(1
           1)@2001-01-03 08:00:00+02)", "[POINT(1
           1)@2001-01-04 08:00:00+02, POINT(2 2)@2001-01-05
           08:00:00+02,POINT(3 3)@2001-01-06 08:00:00+02]"
        };
TGeogPointSeqSet tGeogPointSeqSet = new
   TGeogPointSeqSet(sequences);
```

3. The string array and stepwise constructor, that receives **a boolean and an array of valid strings**.

```
// TGeogPointSeqSet(boolean stepwise, String[] values)
String[] sequences = new String[]{
    "Interp=Stepwise;(1.8@2001-01-01 08:00:00+02,
       1.9@2001-01-03 08:00:00+02]",
    "Interp=Stepwise;[2.3@2001-01-04 08:00:00+02,
       3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
       08:00:00+02]"
```

```
    };
TGeogPointSeqSet tGeogPointSeqSet = new TGeogPointSeqSet(true,
    sequences);
```

4. The string array and SRID constructor, that receives **an int and an array of valid strings**.

```
// TGeogPointSeqSet(int srid, String[] values)
 String[] sequences = new String[]{
    "SRID=4326;[POINT(1 1)@2001-01-01 08:00:00+02, POINT(1
       1)@2001-01-03 08:00:00+02)",
    "SRID=4326;[POINT(1 1)@2001-01-04 08:00:00+02, POINT(2
       2)@2001-01-05 08:00:00+02, POINT(3 3)@2001-01-06
       08:00:00+02]"
    };
TGeogPointSeqSet tGeogPointSeqSet = new TGeogPointSeqSet(4326,
    sequences);
```

5. The string array, SRID and stepwise constructor, that receives **an int, a boolean and an array of valid strings**.

```
// TGeogPointSeqSet(int srid, boolean stepwise, String[]
    values)
String[] sequences = new String[]{
    "Interp=Stepwise;(1.8@2001-01-01 08:00:00+02,
       1.9@2001-01-03 08:00:00+02]",
    "Interp=Stepwise;[2.3@2001-01-04 08:00:00+02,
       3.4@2001-01-05 08:00:00+02, 3.5@2001-01-06
       08:00:00+02]"
    };
TGeogPointSeqSet tGeogPointSeqSet = new TGeogPointSeqSet(4326,
    true, sequences);
```

6. The TGeogPointSeq array constructor, that receives **an array of TGeomPointSeq**.

```
// TGeogPointSeqSet(TGeogPointSeq[] values)
TGeogPointSeq[] sequences = new TGeogPointSeq[]{
    new TGeogPointSeq("[POINT(1 1)@2001-01-01 08:00:00+02,
       POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeogPointSeq("[POINT(1 1)@2001-01-04 08:00:00+02,
       POINT(2 2)@2001-01-05 08:00:00+02, POINT(3
       3)@2001-01-06 08:00:00+02])
    }";
TGeogPointSeqSet tGeogPointSeqSet = new
    TGeogPointSeqSet(sequences);
```

7. The TGeogPointSeq array and stepwise constructor, that receives **a boolean and an array of TGeogPointSeq**.

```
// TGeogPointSeqSet(boolean stepwise, TGeogPointSeq[] values)
TGeogPointSeq[] sequences = new TGeogPointSeq[]{
    new TGeogPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-01
       08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeogPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-04
       08:00:00+02, POINT(2 2)@2001-01-05 08:00:00+02,
       POINT(3 3)@2001-01-06 08:00:00+02]")
        };
TGeogPointSeqSet tGeogPointSeqSet = new TGeogPointSeqSet(true,
   sequences);
```

8. The TGeogPointSeq array and SRID constructor, that receives **a int and an array of TGeogPointSeq**.

```
// TGeogPointSeqSet(int srid, TGeogPointSeq[] values)
TGeogPointSeq[] sequences = new TGeogPointSeq[]{
    new TGeogPointSeq("SRID=4326;[POINT(1 1)@2001-01-01
       08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeogPointSeq("SRID=4326;[POINT(1 1)@2001-01-04
       08:00:00+02, POINT(2 2)@2001-01-05 08:00:00+02,
       POINT(3 3)@2001-01-06 08:00:00+02]")
        };
TGeogPointSeqSet tGeogPointSeqSet = new TGeogPointSeqSet(4326,
   sequences);
```

9. The TGeogPointSeq array, SRID and stepwise constructor, that receives **an int, a boolean and an array of TGeogPointSeq**.

```
// TGeogPointSeqSet(int srid, boolean stepwise,
   TGeogPointSeq[] values)
TGeogPointSeq[] sequences = new TGeogPointSeq[]{
    new TGeogPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-01
       08:00:00+02, POINT(1 1)@2001-01-03 08:00:00+02)"),
    new TGeogPointSeq("Interp=Stepwise;[POINT(1 1)@2001-01-04
       08:00:00+02, POINT(2 2)@2001-01-05 08:00:00+02,
       POINT(3 3)@2001-01-06 08:00:00+02]")
        };
TGeogPointSeqSet tGeogPointSeqSet = new TGeogPointSeqSet(4326,
   true, sequences);
```

4.3.4.6 TTextSeqSet

There are 3 constructors available to create an object of the type *TTextSeqSet*:

1. The string constructor, that receives as single argument **a string** with a valid *TTextSeqSet* value.

```
// TTextSeqSet(String value)
TTextSeqSet tTextSeqSet = new TTextSeqSet("{(A@2001-01-01
   08:00:00+02, B@2001-01-03 08:00:00+02], [C@2001-01-04
   08:00:00+02, D@2001-01-05 08:00:00+02, E@2001-01-06
   08:00:00+02]}"
```

2. The string array constructor, that receives **an array of valid strings**.

```
// TTextSeqSet(String[] values)
String[] stringSequences = new String[]{
    "(A@2001-01-01 08:00:00+02, B@2001-01-03 08:00:00+02]",
    "[C@2001-01-04 08:00:00+02, D@2001-01-05 08:00:00+02,
       E@2001-01-06 08:00:00+02]"
    };
TTextSeqSet tTextSeqSet = new TTextSeqSet(sequences);
```

3. The TTextSeq array constructor, that receives **an array of TTextSeq**.

```
// TTextSeqSet(TTextSeq[] values)
TTextSeq[] sequences = new TTextSeq[]{
        new TTextSeq("(A@2001-01-01 08:00:00+02, B@2001-01-03
           08:00:00+02]"),
        new TTextSeq("[C@2001-01-04 08:00:00+02, D@2001-01-05
           08:00:00+02, E@2001-01-06 08:00:00+02]")
        };
TTextSeqSet tTextSeqSet = new TTextSeqSet(sequences);
```

Chapter 5

Development Environment

5.1 Requirements

The requirements for the *MobilityDB-JDBC* are:

- Java >=11
- Docker

5.2 Build

MobilityDB-JDBC is using:

- Gradle[1] for building
- Maven[2] for the dependencies

Neither Gradle nor Maven need to be installed, it is possible to build the project by running the following command on the root folder.

- For Linux:

```
./gradlew build
```

- For Windows:

```
gradlew.bat build
```

To only run the unit tests, run the following command.

- For Linux:

```
./gradlew test
```

- For Windows:

```
gradlew.bat test
```

[1] https://gradle.org/
[2] https://maven.apache.org/

5.3 Code Analysis

To run the code analysis is required to have a project configured on SonarQube and then execute the command:

- For Linux:

```
./gradlew sonarqube -Dsonar.projectKey={project key}
    -Dsonar.host.url={host} -Dsonar.login={token}
```

- For Windows:

```
gradlew.bat sonarqube -Dsonar.projectKey={project key}
    -Dsonar.host.url={host} -Dsonar.login={token}
```

5.3.1 SonarQube Docker Image

To configure the SonarQube project it is recommended to run the docker image. For more details check the SonarQube setup guide.[3]

First download the docker image:

```
docker pull sonarqube
```

Then run the image for the first time:

```
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p
    9000:9000 sonarqube:latest
```

After that, run the image after a restart:

```
sudo docker start sonarqube
```

Once the image is running open http://localhost:9000 in a browser and login using:

- User: admin
- Password: admin

Create a new project and:

1. Set project key and display name e.g MobilityDB-JDBC
2. Select the option to analyze locally
3. Generate a token e.g 1234
4. Select Gradle for the option that describes the build
5. Copy the gradle command and execute it. For example:

```
./gradlew sonarqube -Dsonar.projectKey=MobilityDB-JDBC
    -Dsonar.host.url=http://localhost:9000 -Dsonar.login=1234
```

6. Review the results on SonarQube

[3] https://docs.sonarqube.org/latest/setup/get-started-2-minutes/

5.4 Running Integration Tests

To run the integration tests is required to have a PostgreSQL database with MobilityDB extension. The integration tests uses the connection string *jdbc:postgresql://localhost:25432/mobilitydb*, but it can be modified in the *BaseIntegrationTest* class. To execute them, run the following command:
For Linux:

```
./gradlew integrationTests
```

For Windows:

```
gradlew.bat integrationTests
```

5.5 MobilityDB Docker Image

It is recommended to run the MobilityDB docker image for the integration tests. For more details check the repository of MobilityDB in the section Docker Image[4]

```
docker pull mobilitydb/mobilitydb
docker volume create mobilitydb_data
docker run --name "mobilitydb" -d -p 25432:5432 -v
    mobilitydb_data:/var/lib/postgresql mobilitydb/mobilitydb
```

[4]https://github.com/MobilityDB/MobilityDBdocker-container

Chapter 6

Running the Example

The workshop example is based on *MobilityDB workshop*[6] for showing the use of MobilityDB. It uses **MobilityDB-JDBC** to create the ship rows and reads them to set the trajectories.

The Workshop example has 6 execution steps:

- **createSchema**, that creates the required tables.
- **loadFromCSV**, that loads the workshop data to *AISInput* table.
- **cleanupData**, that cleans up *AISInput* table.
- **filterData**, that filters the data that is outside a defined window and then saves it to *AISInputFiltered*
- **loadShipData**, loads the ship data using MobilityDB-JDBC instead of a query.

```
private static void loadShipData(Connection con) throws
    SQLException {
    ...
    String sql = "SELECT MMSI, T, ST_Transform(Geom, 25832),
        SOG, COG " +
                 "FROM AISInputFiltered " +
                 "ORDER BY MMSI, T; ";
    Statement readStatement = con.createStatement();
    ResultSet rs = readStatement.executeQuery(sql);
    boolean first = false;
    int currentMmsi = 0;
    List<TGeomPointInst> pointList = new ArrayList<>();
    List<TFloatInst> sogList = new ArrayList<>();
    List<TFloatInst> cogList = new ArrayList<>();

    while (rs.next()) {
```

```
        int mmsi = rs.getInt(1);

        if (!first) {
            currentMmsi = mmsi;
        }

        if (mmsi != currentMmsi) {
            saveShip(con, currentMmsi, pointList, sogList,
                cogList);
            currentMmsi = mmsi;
            pointList = new ArrayList<>();
            sogList = new ArrayList<>();
            cogList = new ArrayList<>();
        }

        OffsetDateTime time = rs.getObject(2,
            OffsetDateTime.class);
        PGgeometry pgGeometry = (PGgeometry) rs.getObject(3);
        pointList.add(new
            TGeomPointInst((Point)pgGeometry.getGeometry(),
            time));
        float sog = rs.getFloat(4);

        if (!rs.wasNull()) {
            sogList.add(new TFloatInst(sog, time));
        }

        float cog = rs.getFloat(5);

        if (!rs.wasNull()) {
            cogList.add(new TFloatInst(cog, time));
        }

        first = true;
    }
    saveShip(con, currentMmsi, pointList, sogList, cogList);
    readStatement.close();
    ...
}

private static void saveShip(Connection con, int mmsi,
                             List<TGeomPointInst>
                                pointList,
                             List<TFloatInst> sogList,
                             List<TFloatInst> cogList)
                                throws SQLException {
    TGeomPointSeq pointSeq = new
```

```
        TGeomPointSeq(pointList.toArray(new
        TGeomPointInst[0]));
    TFloatSeq sogSeq = new TFloatSeq(sogList.toArray(new
        TFloatInst[0]));   TFloatSeq cogSeq = new
        TFloatSeq(cogList.toArray(new TFloatInst[0]));

    PreparedStatement insertStatement = con.prepareStatement(
                "INSERT INTO ships( " +
                        "mmsi, trip, sog, cog) " +
                        "VALUES (?, ?, ?, ?);");
    insertStatement.setInt(1, mmsi);
    insertStatement.setObject(2, new TGeomPoint(pointSeq));
    insertStatement.setObject(3, new TFloat(sogSeq));
    insertStatement.setObject(4, new TFloat(cogSeq));
    insertStatement.execute();
    insertStatement.close();
}

private static void updateShipTrajectory(Connection con, int
    mmsi, TGeomPoint tGeomPoint) throws SQLException {
    PreparedStatement insertStatement = con.prepareStatement(
            "UPDATE ships " +
                "SET traj=trajectory(?) " +
                "WHERE mmsi=?;");
    insertStatement.setObject(1, tGeomPoint);
    insertStatement.setInt(2, mmsi);
    insertStatement.execute();
    insertStatement.close();
}
```

- **setTrajectory**, sets the trajectory using MobilityDB-JDBC instead of a query.

```
private static void setTrajectory(Connection con) throws
    SQLException {
    ...
    String sql = "SELECT MMSI, TRIP FROM SHIPS;";
    Statement readStatement = con.createStatement();
    ResultSet rs = readStatement.executeQuery(sql);

    while (rs.next()) {
        int mmsi = rs.getInt(1);
        TGeomPoint tGeomPoint = (TGeomPoint) rs.getObject(2);
        updateShipTrajectory(con, mmsi, tGeomPoint);
    }

    readStatement.close();
    ...
}
```

6.1 Preparation

To execute this example, it is required to have the workshop docker image:

```
docker pull mobilitydb/mobilitydb:12-2.5-develop-workshop
docker volume create mobilitydb_data_workshop
docker run --name "mobilitydb_workshop" -d -p 25433:5432 -v
    mobilitydb_data_workshop:/var/lib/postgresql
    mobilitydb/mobilitydb:12-2.5-develop-workshop
```

We will need to create a database with the name *DanishAIS* that has the MobilityDB extension.

```
CREATE EXTENSION MobilityDB CASCADE;
```

6.2 Execution

Then you need to go in the command line to the root folder of the project, and execute the following command

- For Linux:

  ```
  ./gradlew -PmainClass=com.mobilitydb.example.Workshop run
  ```

- For Windows:

  ```
  gradlew.bat -PmainClass=com.mobilitydb.example.Workshop run
  ```

The summary of time in ms per execution step will be shown in the terminal. In the following table 6.1, we can see the times when running in a virtual Linux machine and in a real Windows machine.

Execution step	***Linux***	***Windows***	***Uses MobilityDB-JDBC***
Create Schema	309ms	41ms	
Load From CSV	16881ms	10477ms	
Cleanup Data	29716ms	13644ms	
Filter Data	24107ms	27312ms	
Load Ship Data	36480ms	19949ms	x
Set Trajectory	21019ms	8065ms	x

Table 6.1: Comparison of times when runing in Linux virtual machine and in Windows

Table	***Rows***
AISInput	1209962
AISInputFiltered	601945
ships	67

Table 6.2: Number of records by table

6.3 Drawing the Results in QGIS

First of all we need to have QGIS installed[1].
Then we need to load the map where we the trajectories are going to be shown, the figure 6.1 shows how to load the map.

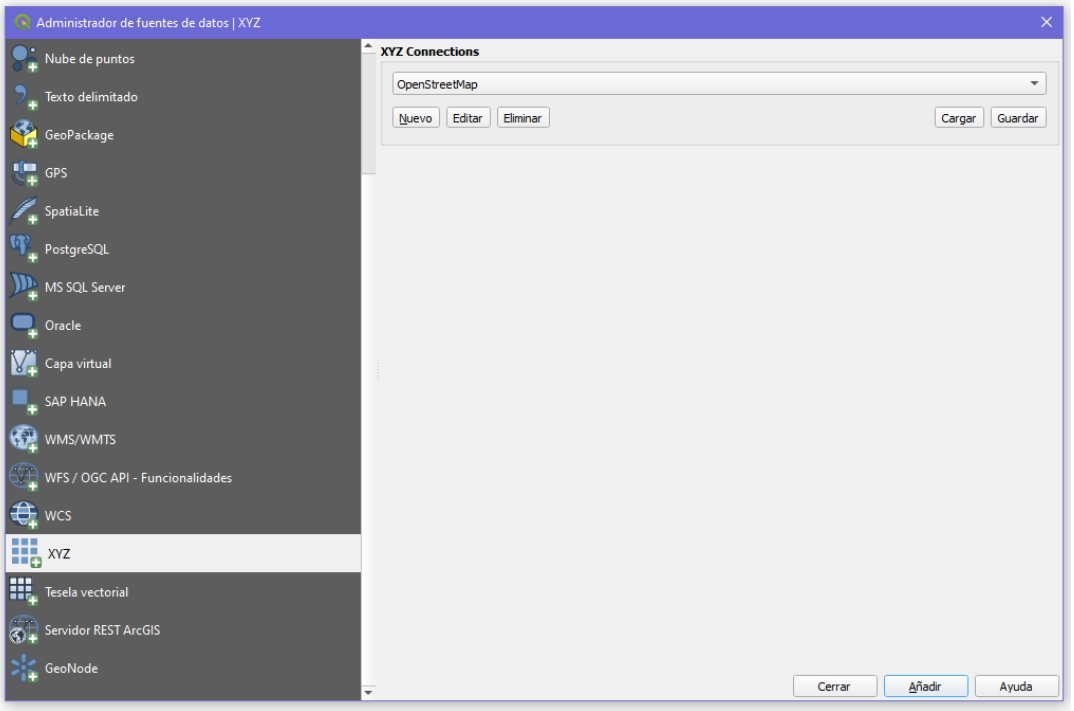

Figure 6.1: QGis: Load the map

After having loaded the map, we need to create a connection to the database, We need to set the name (any name), the host (in this case *localhost*), the database (*DanishAIS*) and user and password (*docker/docker*) The figure 6.2 shows how to create the connection.

[1]The installer could be dowonloaded from https://www.qgis.org/en/site/forusers/download.html

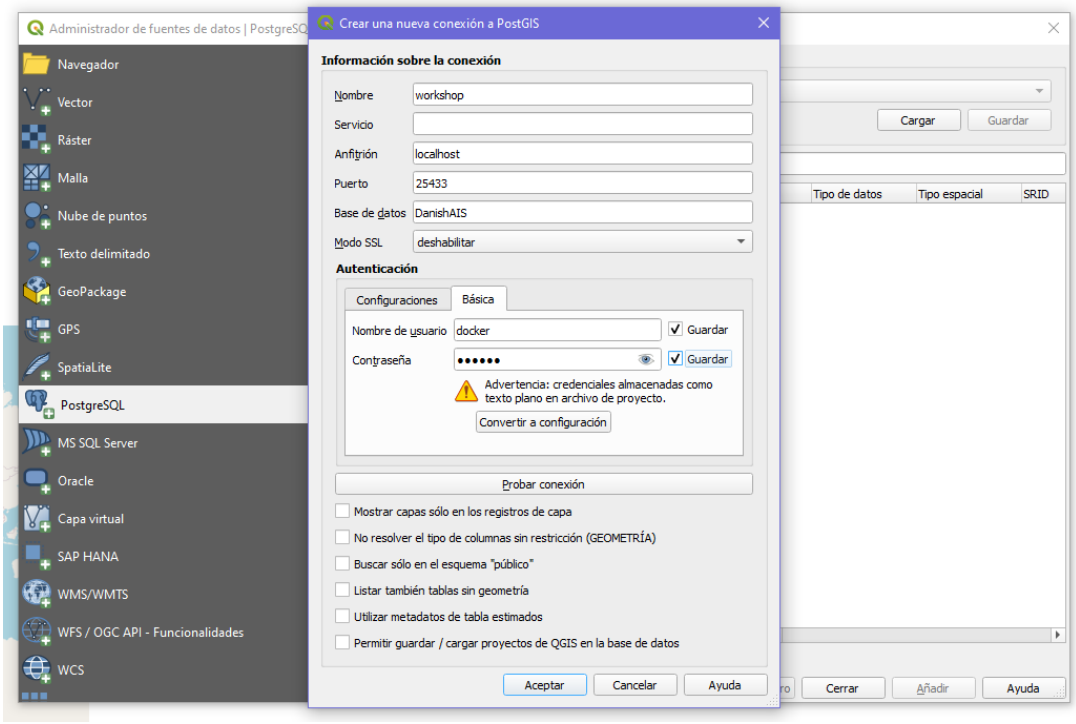

Figure 6.2: QGis: Create the connection to PostgreSQL

Then we need to press *Connect* and then select which table (*ships*) has the data to draw in the map. The figure 6.3 shows how to create the connection.

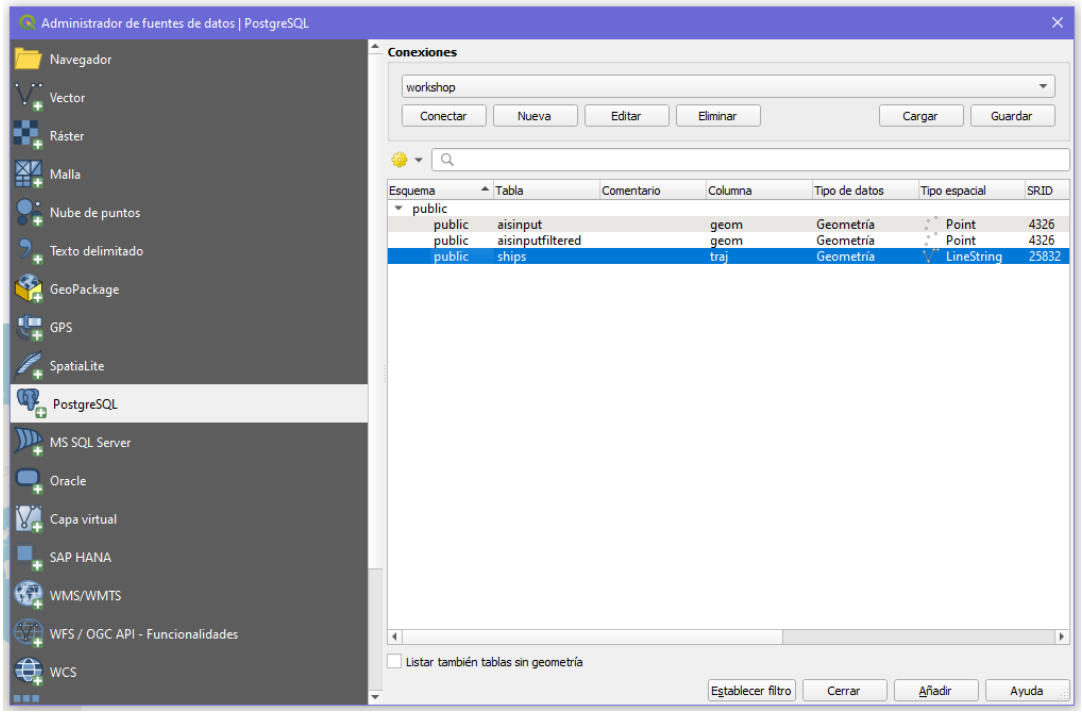

Figure 6.3: QGis: Select the table from the database

QGIS will show the map with trajectories drawn in orange, as shown in figure 6.4.

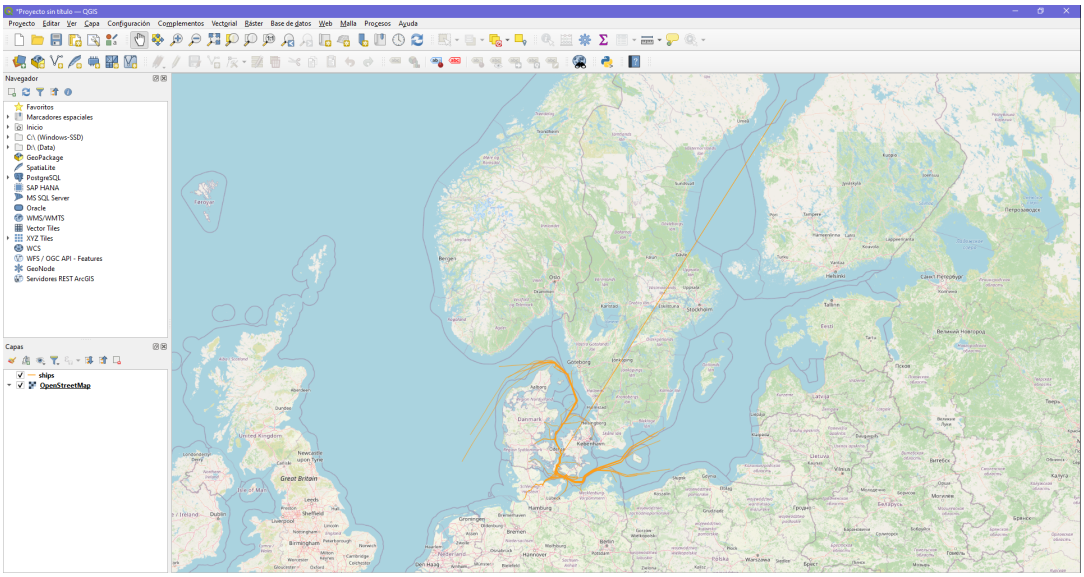

Figure 6.4: QGis: Trajectories drawn in the map.

Chapter 7

Conclusions

- The initial design for the Time Types and Box Types, was enough for developing the Temporal types, and extending it for their requirements, taking advantage of *generics, inheritance and polymorphism*
- The unit test and integration tests were useful for verifying that the modules were working as expected independently and together, and when there were code refactors they helped to check if any code change was affecting the previous functionality.
- Working with the example gave a more realistic view of how to use the MobilityDB-JDBC driver in a development environment.
- SonarQube helped with the code maintainability and quality, reducing possible bugs and duplications.
- There is feature parity with the MobilityDB Python Adapter, although they are similar in functionality, the way of using is different because Python is an untyped language and Java a typed language, this is more evident when we take a look at the number of constructors for some MobilityDB types.

Appendices

Annex A

Example of PostGIS JDBC Usage

```
import java.sql.*;
import java.util.*;
import java.lang.*;
import org.postgis.*;

public class JavaGIS {

public static void main(String[] args) {

  java.sql.Connection conn;

  try {
    /*
    * Load the JDBC driver and establish a connection.
    */
    Class.forName("org.postgresql.Driver");
    String url = "jdbc:postgresql://localhost:5432/database";
    conn = DriverManager.getConnection(url, "postgres", "");
    /*
    * Add the geometry types to the connection. Note that you
    * must cast the connection to the pgsql-specific connection
    * implementation before calling the addDataType() method.
    */
    ((org.postgresql.PGConnection)conn).addDataType("geometry",Class.forName("org.postgis.PGgeometry"));
    ((org.postgresql.PGConnection)conn).addDataType("box3d",Class.forName("org.postgis.PGbox3d"));

    /*
    * Create a statement and execute a select query.
    */
    Statement s = conn.createStatement();
    ResultSet r = s.executeQuery("select geom,id from geomtable");
    while( r.next() ) {
      /*
      * Retrieve the geometry as an object then cast it to the geometry type.
      * Print things out.
      */
      PGgeometry geom = (PGgeometry)r.getObject(1);
      int id = r.getInt(2);
      System.out.println("Row " + id + ":");
      System.out.println(geom.toString());
    }
    s.close();
    conn.close();
  }
catch( Exception e ) {
  e.printStackTrace();
  }
}
}
```

Figure A.1: Usage of PostGIS JDBC [10]

Annex B

Methods by DataType

	Time			Boxes	
Methods/ DataTypes	**TimestampSet**	**Period**	**PeriodSet**	**TBox**	**STBox**
setValue()	x	x	x	x	x
getValue()	x	x	x	x	x
contains(OffsetDatetime)		x			
overlap(Period)		x			
getValues()					
startValue()					
endValue()					
minValue()					
maxValue()					
valueAtTimestamp(OffsetDateTime)					
getTime()					
duration()		x	x		
timespan()	x		x		
period()	x		x		
numPeriods()			x		
startPeriod()			**x**		
endPeriod()			**x**		
periodN(int)			**x**		
periods()			x		
numInstants()					
startInstant()					
endInstant()					
instantN(int)					
instants()					
numTimestamps()	x		x		
startTimestamp()	x		x		
endTimestamp()	x		x		
timestampN(int)	x		x		
timestamps()	x		x		
shift(Duration)	x	x	x		
intersectsTimestamp(OffsetDateTime)					
intersectsPeriod(Period)					
getTimestamp()					
numSequences()					
startSequence()					
endSequence()					
sequenceN(int)					
sequences()					

Table B.1: Methods by data type - Time and Box types

	Temporal			
Methods/ DataTypes	**TInst**	**TInstSet**	**TSeq**	**TSeqSet**
setValue()				
getValue()	x			
contains(OffsetDatetime)				
overlap(Period)				
getValues()	x	x	x	x
startValue()	x	x	x	x
endValue()	x	x	x	x
minValue()	x	x*	x*	x*
maxValue()	x	x*	x*	x*
valueAtTimestamp(OffsetDateTime)	x	x	x	x
getTime()	x	x	x	x
duration()	x	x	x	x
timespan()	x	x	x	x
period()	x	x	x	x
numPeriods()				
startPeriod()				
endPeriod()				
periodN(int)				
periods()				
numInstants()	x	x	x	x
startInstant()	x	x	x	x
endInstant()	x	x	x	x
instantN(int)	x	x	x	x
instants()	x	x	x	x
numTimestamps()	x	x	x	x
startTimestamp()	x	x	x	x
endTimestamp()	x	x	x	x
timestampN(int)	x	x	x	x
timestamps()	x	x	x	x
shift(Duration)	x	x	x	x
intersectsTimestamp(OffsetDateTime)	x	x	x	x
intersectsPeriod(Period)	x	x	x	x
getTimestamp()	x			
numSequences()			x	x
startSequence()			x	x
endSequence()			x	x
sequenceN(int)			x	x
sequences()			x	x

* *These methods were not implemented for TGeogPoint and TGeomPoint, because there wasn't a way of compare two points and determine the minimum and the maximum.*

Table B.2: Methods by data type - Temporal types

Annex C

Methods Definition

Method	***Definition***
setValue()	Sets the value to the object
getValue()	Gets the value
contains(OffsetDatetime)	If the OffsetDatetime is contained in the period
overlap(Period)	If there any timestamp in common
getValues()	Gets the values
startValue()	Gets the first value
endValue()	Gets the last value
minValue()	Gets the minimum value
maxValue()	Gets the maximum value
valueAtTimestamp(OffsetDateTime)	Gets the value at the timestamp sent
getTime()	"Gets the periodset on which the temporal value is defined"[45]
duration()	"Gets the interval on which the temporal value is defined"[45]
timespan()	"Gets the interval on which the temporal value is defined ignoring the potential time gaps"[45]
period()	Gets the period
numPeriods()	Gets the number of periods
startPeriod()	Gets the first period
endPeriod()	Gets the last period
periodN(int)	Gets the period locate in the index sent
periods()	Get all periods
numInstants()	Gets the number of instants
startInstant()	Gets the first instant
endInstant()	Gets the last instant
instantN(int)	Gets the period located at the index position
instants()	Gets all instants
numTimestamps()	Gets the number of timestamps
startTimestamp()	Gets the first timestamp
endTimestamp()	Gets the last instant
timestampN(int)	Gets the timestamps located at the index position
timestamps()	Gets all timestamps
shift(Duration)	Shifts the duration sent
intersectsTimestamp(OffsetDateTime)	If the temporal value intersects the OffsetDatetime sent
intersectsPeriod(Period)	If the temporal value intersects the Period sent
getTimestamp()	Gets the timestamp
numSequences()	Gets the number of sequences
startSequence()	Gets the first sequence
endSequence()	Gets the last sequence
sequenceN(int)	Gets the sequence located at the index position
sequences()	Gets all sequences

Table C.1: Methods definition

Bibliography

[1] Junaid Ahmed. *JUnit vs. TestNG: Choosing a Framework for Unit Testing.* 2019. URL: https://www.stickyminds.com/article/junit-vs-testng-choosing-framework-unit-testing (visited on 06/16/2022).

[2] Mark Altaweel. *What is PostGIS?* 2017. URL: https://www.gislounge.com/what-is-postgis/.

[3] The asyncpg authors and contributors. *asyncpg – A fast PostgreSQL Database Client Library for Python/asyncio.* https://github.com/MagicStack/asyncpg. 2020.

[4] Software Testing Help blog. *JUnit Tutorial For Beginners – What Is JUnit Testing.* URL: https://www.softwaretestinghelp.com/junit-tutorial/#What_Is_Unit_Testing (visited on 07/18/2022).

[5] Université libre de Bruxelles and MobilityDB contributors. *MobilityDB.* https://github.com/MobilityDB/MobilityDB. 2022.

[6] Université libre de Bruxelles and MobilityDB contributors. *MobilityDB Workshop.* https://github.com/MobilityDB/MobilityDB-workshop. 2021.

[7] Université libre de Bruxelles and MobilityDB contributors. *MobilityDB-python.* https://github.com/MobilityDB/MobilityDB-python. 2021.

[8] TIOBE Software BV. *TIOBE Index for January 2022.* 2022. URL: https://www.tiobe.com/tiobe-index/.

[9] Pierre Carbonnelle. *PYPL PopularitY of Programming Language.* 2022. URL: http://pypl.github.io/PYPL.html.

[10] *Chapter 5. Using PostGIS: Building Applications.* PostGIS. URL: https://postgis.net/docs/manual-1.5/ch05.html (visited on 08/08/2022).

[11] PostGIS Project Steering Committee. *PostGIS - Spatial and Geographic objects for PostgreSQL.* URL: https://postgis.net/.

[12] PostGIS Project Steering Committee. *postgis-java.* URL: https://github.com/postgis/postgis-java/blob/master/jdbc/README.

[13] Microsoft Company. *Citus: Distributed PostgreSQL as an extension*. https://github.com/citusdata/citus. 2022.

[14] Oracle Corporation. *Coordinate Systems (Spatial Reference Systems)*. URL: https://docs.oracle.com/cd/A91202_01/901_doc/appdev.901/a88805/sdo_cs_c.htm.

[15] Mark Drake and Ostezer. *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. URL: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[16] EDUCBA. *What is PostgreSQL?* URL: https://www.educba.com/what-is-postgresql/.

[17] Mahmoud Sakr Esteban Zimányi Arthur Lesuisse and Mohamed Bakli. *MobilityDB: A Mainstream Moving Object Database System*. https://docs.mobilitydb.com/pub/MobilityDBDemo_SSTD19.pdf. 2019.

[18] The Psycopg Team. Federico Di Gregorio Daniele Varrazzo. *psycopg2 - Python-PostgreSQL Database Adapter*. https://github.com/psycopg/psycopg2. 2020.

[19] Avipsha Ghosh. *Unit, Integration, and Functional Testing: 4 main points of difference*. 2022. URL: https://www.headspin.io/blog/unit-integration-and-functional-testing-4-main-points-of-difference.

[20] Anita Graser and collaborators. *MovingPandas*. https://github.com/anitagraser/movingpandas/. 2020.

[21] The PostgreSQL Global Development Group. *PostgreSQL JDBC Driver*. URL: https://jdbc.postgresql.org/index.html.

[22] The PostgreSQL Global Development Group. *PostgreSQL: Documentation*. URL: https://www.postgresql.org/docs/.

[23] The PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. URL: https://www.postgresql.org.

[24] The PostgreSQL Global Development Group. *The PostgreSQL JDBC Interface*. URL: https://jdbc.postgresql.org/documentation/head/index.html.

[25] *Introduction to Javadoc*. Baeldung. 2022. URL: https://www.baeldung.com/javadoc.

[26] *Javadoc*. Oracle. URL: https://docs.oracle.com/javase/9/javadoc/javadoc.htm (visited on 08/08/2022).

[27] Adonmo Private Limited. *MobilityDB SQLAlchemy*. https://github.com/adonmo/mobilitydb-sqlalchemy. 2020.

[28] Aurelio Morales. *MobilityDB: extensión de PostGIS para objetos en movimiento*. 2019. URL: https://mappinggis.com/2019/10/mobilitydb-extension-de-postgis-para-objetos-en-movimiento/.

[29] Nishadha. *UML Class Diagram Relationships Explained with Examples*. 2022. URL: https://creately.com/blog/diagrams/class-diagram-relationships/.

[30] P. Pankaj and A. Saini. *Difference between Unit Testing and Integration Testing*. 2022. URL: https://www.geeksforgeeks.org/difference-between-unit-testing-and-integration-testing/.

[31] Official JDBC driver for PostgreSQL. *PostgreSQL JDBC Driver*. https://github.com/pgjdbc/pgjdbc. 2022.

[32] *PostgreSQL Extensions to the JDBC API*. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/7.4/jdbc-ext.html (visited on 08/08/2022).

[33] *PostgreSQL JDBC Driver*. The PostgreSQL Global Development Group. URL: https://jdbc.postgresql.org/about/about.html (visited on 08/06/2022).

[34] SonarSource S.A. *SonarQube Documentation*. URL: https://docs.sonarqube.org/latest/.

[35] Lisa Smith. *What PostgreSQL has over other open source SQL databases: Part I*. URL: https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases.

[36] *Spatial reference system*. Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/wiki/Spatial_reference_system (visited on 07/30/2022).

[37] Johannes Link Stefan Bechtold Sam Brannen and collaborators. *JUnit 5 User Guide*. URL: https://junit.org/junit5/docs/current/user-guide/#overview.

[38] *The Differences Between Unit Testing, Integration Testing And Functional Testing*. SOFTWARETESTINGHELP.COM. 2022. URL: https://www.softwaretestinghelp.com/the-difference-between-unit-integration-and-functional-testing/.

[39] tilery. *python-postgis*. https://github.com/tilery/python-postgis. 2019.

[40] PostgreSQL Tutorial Website. *What is PostgreSQL?* URL: https://www.postgresqltutorial.com/what-is-postgresql/.

[41] *What is an SRID?* Esri ArcGIS. URL: https://desktop.arcgis.com/en/arcmap/latest/manage-data/using-sql-with-gdbs/what-is-an-srid.htm (visited on 07/30/2022).

[42] *World Geodetic System (WGS84)*. GIS Geography. URL: https://gisgeography.com/wgs84-world-geodetic-system/ (visited on 08/08/2022).

[43] Esteban Zimanyi. *MobilityDB 1.0 User's Manual*. 2021. URL: https://docs.mobilitydb.com/MobilityDB/develop/mobilitydb-manual.pdf.

[44] Esteban Zimanyi. *MobilityDB: A PostgreSQL Extension for Mobility Data Management*. PGConf.Russia. 2019. URL: https://pgconf.ru/en/2019/242944.

[45] Esteban Zimanyi. *Welcome to python-mobilitydb's documentation!* 2020. URL: https://docs.mobilitydb.com/python-mobilitydb/master/.

[46] Esteban Zimanyi and Mahmoud Sakr. *MobilityDB: Managing Mobility Data in PostgreSQL*. PGConf.Russia. 2020. URL: https://pgconf.ru/en/2020/264545.