MARCOS MIGUEL PÁSCOA PARREIRA ROSA

DESENVOLVIMENTO DE BASE DE DADOS ESPACIAIS PARA DADOS BATIMÉTRICOS E ALTIMÉTRICOS DE ELEVADA RESOLUÇÃO

UNIVERSIDADE DO ALGARVE
Faculdade de Ciências e Tecnologia
2019

MARCOS MIGUEL PÁSCOA PARREIRA ROSA

DESENVOLVIMENTO DE BASE DE DADOS ESPACIAIS PARA DADOS BATIMÉTRICOS E ALTIMÉTRICOS DE ELEVADA RESOLUÇÃO

Mestrado em Geomática

Trabalho efectuado sob a orientação de:

Doutor Joaquim Luís

Doutor Nuno Lourenço

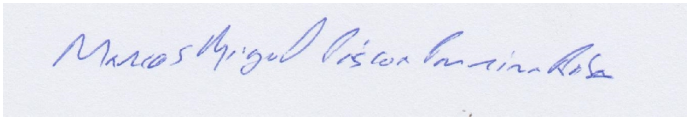
UNIVERSIDADE DO ALGARVE
Faculdade de Ciências e Tecnologia
2019

DESENVOLVIMENTO DE BASE DE DADOS ESPACIAIS PARA DADOS BATIMÉTRICOS E ALTIMÉTRICOS DE ELEVADA RESOLUÇÃO

DECLARAÇÃO DE AUTORIA DO TRABALHO

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Marcos Miguel Páscoa Parreira Rosa



…….....…

AGRADECIMENTOS

Deixo aqui expresso um agradecimento aos professores do mestrado, por todo o conhecimento transmitido e palavras de motivação. Aos professores Joaquim Luís e Nuno Lourenço, pela disponibilidade em serem meus orientadores, pelos incentivos, discussões e ajuda na realização deste trabalho. Agradeço igualmente aos colegas da Divisão de Geologia e GeoRecursos Marinhos do Instituto Português do Mar e da Atmosfera por todo o apoio dado.

RESUMO

O presente projeto pretende implementar uma solução prática e funcional para armazenamento e consulta eficazes de dados de batimetria e altimetria de elevada resolução e respectivos metadados, com recurso a software livre. Foi escolhido o sistema de bases de dados relacional PostgreSQL, complementado com o conjunto de funções espaciais designado por PostGIS e ainda por livraria de funções ("*pgpointcloud*") para utilização de estruturas de dados tipo "*Point Cloud*" (nuvens de pontos).

Para a gestão da informação, pesquisas com critérios espaciais e visualização dos dados foi desenvolvido programa utilizando a linguagem Python (versão 2.7), complementada com a infraestrutura de interface gráfica (*GUI*) Qt 4 e ainda com a livraria Qgis para visualização de mapas. Foi construída uma base de dados de teste, carregada com diversos conjuntos de dados procedentes de levantamentos batimétricos do tipo multi-feixe e feixe simples.

Os dados foram armazenados segundo uma estrutura do tipo *"PointCloud Point"*, em que cada ponto é representado por um registo que agrega as coordenadas x e y (latitude e longitude WGS84), profundidade/altitude (metros), valor de retrodispersão ("*backscatter*") e um índice RGB. Os dados a integrar (organizados por cruzeiro/missão) são convertidos para "*PC Points*" e agrupados segundo células rectangulares com um número máximo de 600 pontos. Cada célula ("*patch*") inclui informação espacial e ocupa uma única linha na tabela correspondente na base de dados, permitindo grande eficácia no manuseamento e visualização de dados. Foram criadas tabelas de metadados de acordo com os vocabulários definidos pelo programa SeaDataNet da União Europeia.

Foram implementadas funcionalidades de criação de base de dados de forma automatizada, importação de dados a partir de ficheiros de texto (x,y,z), conversão de sistemas de coordenadas, pesquisa por critérios geográficos e/ou por atributos e exportação de dados espaciais em formato texto ou "shapefile" e respectivos metadados. Todo o sistema foi desenvolvido exclusivamente com recurso a software livre ("*FOSS*"), permitindo a implementação da base de dados tanto em Windows (8 e 10) como em Linux (testado nas distribuições Lubuntu 14.04.5 e Lubuntu 16.04.3). Não foram efectuados testes para o sistema operativo OSX (Apple), mas não existem, à partida, motivos para não se poder implementar também neste sistema.

Palavras- chave - batimetria, altimetria, base de dados espacial , PostGis, Python, Point Cloud

ABSTRACT

This project aims to implement a practical and functional solution for storing and accessing high resolution bathymetric and topographic data in an efficient way, using free software (FOSS). For data storage and management the PostgreSQL (version 9.3) relational database system was choosed, complemented by PostGIS extensions for spatial data, and the "pgpointcloud" library extension for dealing with point cloud data.

To manage data ingestion, metadata creation, queries by spatial and/or non-spatial attributes and data visualization, a program was developed in the Python sofware language (version 2.7) , making use of Qt4 application framework for GUI purposes (via PyQt libraries) and QGis developer libraries for mapping and data visualization.

A test database was populated with bathymetric (multi beam and single beam). Using the pgpointcloud library, the data points were converted to Point Cloud type (PC Points), with x,y coordinates (WGS84 latitude and longitude), depth/altitude (m), backscatter signal strength, and a RGB index. These points were further assembled in sets of rectangular cells ("patches") up to a maximum of 600 PC Points per cell, indexed by a cruise/mission unique id. Each cell or "patch" includes spatial information and is stored as a single line in the corresponding database table, allowing for efficient data management.

Metadata auxiliary tables were created from the vocabularies defined by the SeaDataNet EU program, and integrated in the test database, allowing the correct data classification in accordance to Emodnet Bathymetry standards.

A set of different functions was developed within the main program to allow for automatic database creation, data ingestion from xyz text files, geographic coordinates transformation, geospatial and attribute queries, spatial data export in text or shapefile formats and metadata export to XML files.

The project was developed in its entirety using Free Software, and can be used both in Windows (8 & 10) and Linux (tested in Lubuntu 14.04.5 and Lubuntu 16.04.3) operating systems. It was not tested for OSX (Apple) but there are no "a priori" reasons for it not to be compatible with this system.

Key-words: altimetry bathymetry, spatial data base, PostGIS, Python, Point Cloud

ÍNDICE

TABELA DE FIGURAS

1. APRESENTAÇÃO

1.1. Escolha do tema do projeto de Mestrado

A escolha de um projeto de conceção de base de dados espaciais para conclusão do Mestrado em Geomática surge na sequência do trabalho desenvolvido no Instituto Português do Mar e da Atmosfera (IPMA), no âmbito do projeto Emodnet Bathymetry (integrado na rede "*European Marine Observation and Data Network*" da Comissão Europeia) do qual o IPMA é parceiro.

Como responsável pela integração de dados de batimetria em duas infra-estruturas pan-europeias de divulgação e partilha de dados ("*CDI Data Discovery and Access*" e "*Sextant Catalogue*") e dada a necessidade de organizar os dados de batimetria existentes no IPMA, prevendo-se ainda a aquisição de mais dados com o sistema multifeixe adquirido em 2015, foi decidido, em conjunto com o Doutor Pedro Terrinha, chefe da Divisão de Geologia Marinha e Geo-Recursos, iniciar o desenvolvimento de uma base de dados espacial que permitisse a agregação de toda a informação de batimetria disponível na instituição.

Após discussão com os coodenadores do Mestrado em Geomática Doutora Cristina Veiga Pires e Doutor José Rodrigues, foi aceite por estes a proposta acima descrita para projeto de Mestrado, tendo sugerindo que fossem os Doutores Joaquim Luís e Nuno Lourenço os orientadores do projeto.

Foi neste quadro, e de forma a potenciar o conhecimento adquirido durante a frequência do Mestrado em Geomática na implementação de uma solução para uma necessidade real do IPMA que surgiu o presente projeto de mestrado.

1.2. Problemática do projeto

A opção de integrar dados de levantamentos batimetricos numa base de dados espacial, por oposição a apenas se arquivarem os dados num conjunto de ficheiros, apresenta várias vantagens. Em primeiro lugar, permite centralizar e sistematizar num único local toda a informação relevante, evitando a dispersão dos dados por várias pastas em diferentes suportes físicos, situação que leva frequentemente a dificuldades na localização de dados e

à existência de dúvidas quanto à origem e estado de processamento dos mesmos. Por outro lado, torna-se possível utilizar ferramentas incluidas no sistema de base de dados para assegurar a integridade dos dados, constituir índices para pequisas rápidas e proceder a consultas sobre a totalidade de uma região abarcando distintas campanhas de aquisição de dados. Permite ainda, após correcto preenchimento dos metadados, proceder a pesquisas baseadas em atributos, como por exemplo identificar dados adquiridos com um determinado equipamento, averiguar qual o tipo de controlo de qualidade utilizado e verificar a existência de dados complementares (medições da velocidade do som na água, dados de retrodispersão acústica do fundo e outros).

1.3. Tipos de dados a integrar

Originalmente, o objectivo do presente projeto era o de se construir uma base de dados apenas com informação batimétrica e respectivos metadados, armazenando valores de profundidades e respectivas coordenadas X e Y num dado sistema geográfico, organizados por cruzeiro/campanha de aquisição de dados. Após discussão com os orientadores do projeto, foi decidido que deveria ser prevista a possibilidade de se considerar também o armazenamento de dados de altimetria obtidos por sistemas LIDAR, nomeadamente para prever a integração dos dados obtidos no levantamento geral da costa do continente promovido pela Direcção Geral do Território em 2011 (que inclui alguma informação de batimetria em zonas pouco profundas), e futuros levantamentos costeiros.

No caso dos dados batimétricos, temos essencialmente dois tipos: dados provenientes de levantamentos com sondas de feixe simples e dados de levantamentos executados com sistemas multi-feixe. Enquanto que no primeiro caso os dados obtidos consistem (regra geral) exclusivamente em valores de profundidade (e respectivas coordenadas x e y) , no segundo caso é frequente ser também adquirida informação de retrodispersão acústica ("*backscatter*") para cada valor de profundidade determinado.

Os dados provenientes de levantamentos LIDAR, para além das coordenadas x e y e respectivo valor de altitude (z) associado, apresentam também, de forma geral, informação quanto à intensidade do sinal reflectido (retrodispersão) e podem ainda apresentar, quando devidamente processados em conjunto com fotografias aéreas, valores RGB.

Dada a analogia com os dados provenientes de sistemas multifeixe, foi prevista a possibilidade de se armazenar este tipo de informação nas estruturas de dados a criar.

1.4. Objectivos do projeto

Foram definidos 5 objectivos principais, que se descrevem abaixo, juntamente com uma breve síntese da sua importância.

1 - Concepção e implementação de estruturas de dados que permitam o armazenamento eficaz dos vários tipos de informação batimétrica e altimétrica disponíveis.

Dado os elevados volumes de dados em questão, pretende-se encontrar formas de optimizar a sua integração em tabelas de um Sistema de Bases de Dados Espaciais (SBDE), bem como assegurar que as estruturas de dados a implementar permitam tirar máximo proveito das ferramentas de análise espacial disponibilizadas pelo sistema escolhido.

2 – Concepção e implementação de conjunto de tabelas auxiliares para integração de metadados.

Um dos pressupostos do projeto é o de permitir a integração com as infra-estruturas pan-europeias de divulgação e partilha de dados com que o IPMA colabora. Desta forma é necessário utilizar os mesmos vocabulários e um conjunto de atributos obrigatórios definidos por essas infra-estruturas para os metadados de cada conjunto de dados a integrar. Esta informação terá que ser integrada em tabelas auxiliares na base de dados de forma a se poderem estabelecer relações entre cada conjunto de dados e os seus respectivos metadados.

3 – Desenvolvimento de aplicação de gestão e manutenção da base de dados que permita a importação de dados e a sua devida integração nas estruturas descritas acima.

Para assegurar que o projeto constitua uma ferramenta útil para a prosecução dos objectivos pretendidos é necessário desenvolver uma aplicação informática que permita a um utilizador sem conhecimentos avançados de funcionamento de bases de dados executar operações fundamentais como a criação de uma base de dados de raíz, importação de conjuntos de dados, preenchimento de atributos (metadados), edição/correcção dos mesmos

e ainda permitir a remoção de conjuntos de dados já integrados que, por algum motivo, se entenda necessário, assegurando a integridade do sistema.

4 - Desenvolvimento de aplicação com interface gráfica para visualização dos dados e construção de pesquisas ("*queries*") espaciais ou por atributos.

Embora o principal objectivo do projeto seja o armazenamento eficaz de dados, de forma a se poder usufruir das vantagens que um SBDE oferece, torna-se praticamente imperativo o desenvolvimento de uma aplicação com interface gráfica ("*GUI*") que permita ao utilizador a visualização da extensão geográfica de cada conjunto de dados, bem como a morfologia associada (batimetria ou altimetria). Esta aplicação deverá ainda possibilitar a execução de pesquisas com diferentes critérios (espaciais ou não) com visualização, em mapa, dos resultados, e possibilidade de os exportar para formatos de utilização comum (ficheiros de texto do tipo xyz, ficheiros GMT, GeoJSON, "*shapefile*").

5 – Desenvolvimento de ferramentas de exportação de dados e metadados para os formatos necessários à integração nas infra-estruturas europeias de divulgação e partilha de dados em que o IPMA participa.

De forma a cumprir com os compromissos do IPMA relativamente à partilha de dados, as funcionalidades referidas acima deverão ser implementadas através de aplicação informática que proceda de forma automatizada à criação dos ficheiros de dados e metadados necessários, após selecção do conjunto de dados (cruzeiro/campanha) pelo utilizador.

1.5. Limites do projeto

O projeto aqui apresentado limita-se a cumprir com as especificações/objectivos acima enunciados. Como acontece na maior parte dos projetos que envolvem programação, prevê-se que após a sua implementação no IPMA, serão necessários novos desenvolvimentos para eliminação de "*bugs*", acréscimo de novas funcionalidades e optimização de processos. Nesta fase pretende-se sobretudo que o projeto assegure as funcionalidades descritas acima, e, que as estruturas de dados implementadas assegurem a integridade dos dados.

Os dados a integrar correspondem a levantamentos devidamente processados e respectivos metadados, não sendo objectivo do projeto assegurar o arquivo de dados em bruto ("*raw data*").

Foi dada prioridade à concepção e implementação de estruturas de dados espaciais que minimizassem o espaço em disco necessário ao armazenamento da informação de batimetria e altimetria, dado o elevado volume de dados resultantes de levantamentos com sistemas multifeixe e LIDAR.

Assim, o desenvolvimento de funcionalidades de visualização de dados (quer em mapa, quer em representação 3D) procurou assegurar um mínimo de "*feedback*" visual ao utilizador, sem se enveredar por procedimentos sofisticados de apresentação gráfica. Note-se que, após integração dos dados no SBDE, estes podem ser explorados por diverso software já existente, nomeadamente Sistemas de Informação Geográfica (SIG), livres ou comerciais.

As entidades espaciais primárias a considerar correspondem a geometrias de tipo Ponto 2.5D, ou seja, pontos no plano x e y, com atributo z correspondente a profundidade ou altitude e ainda, quando dísponíveis nos dados originais, valor de retrodispersão. Não se consideram outros tipos primários de dados espaciais, embora se utilizem, quando adequado, entidades do tipo polígono (visualização dos limites dos conjuntos de dados, generalizações, "*bounding boxes*" para pesquisas espaciais).

1.6. Resumo dos capítulos seguintes

No capítulo Introdução é apresentada uma breve revisão do "estado da arte" quanto à utilização de SBDE para integração de dados de altimetria e batimetria. São ainda descritas as principais fontes de informação utilizadas, organizadas por temas (sistemas de bases de dados espaciais, estruturas de dados tipo PointCloud, índices espacias, infraestruturas de interface gráfica ("*GUI"*) compatíveis com a linguagem Python, livrarias de desenvolvimento QGis e outras livrarias e ferramentas de programação utilizadas.

O capítulo Metodologia apresenta a explicação das estruturas de dados implementadas para armazenamento da informação, diagramas entidade-relação para as diferentes tabelas criadas, estrutura da aplicação desenvolvida, livrarias utilizadas e descrição das suas diferentes funcionalidades.

No capítulo Resultados e Discussão é feita uma análise aos resuldados obtidos, em termos da eficácia da metodologia escolhida para armazenamento de dados e sua adequação a cada um dos 5 objectivos principais do projeto.

O capítulo dedicado às Conclusões pretende constituir uma avaliação global do projeto e dos seus resultados e principais dificuldades encontradas, bem como apresentar propostas de desenvolvimentos futuros.

2. INTRODUÇÃO

2.1. Definição de sistema de base de dados espaciais (SBDE)

Da análise de bibliografia relativa a sistemas de bases de dados espaciais verifica-se, como em outras áreas, diferentes pontos de vista e propostas quanto aos critérios de definição destes sistemas. No entanto, um artigo em particular apresenta uma proposta clara e detalhada de um conjunto de critérios de definição, constituindo um bom ponto de partida para o presente capítulo. Trata-se de contribuição para um número especial da revista "*Very Large Database Systems Journal*", de 1994, da autoria do Prof. Dr. Ralf Hartmut Güting (Universidade de Hagen, Alemanha). Neste artigo, o autor descreve que a necessidade de gerir dados geométricos, geográficos ou, no sentido geral, dados espaciais, em diferentes áreas, como geografia, design industrial, modelação tridimensional, levou a que, desde o aparecimento dos primeiros sistemas de gestão de bases de dados relacionais, se procurasse encontrar formas de armazenar e explorar dados espaciais recorrendo a estes sistemas. O autor salienta como característica distintiva das tecnologias emergentes para dar resposta a estas necessidades, a capacidade de lidar com conjuntos extensos de geometrias relativamente simples, como, por exemplo, um grupo de 100 000 polígonos, estabelecendo no entanto uma distinção com tipos de bases de dados associadas a sistemas de CAD, em que entidades geométricas são organizadas hierárquicamente em estruturas complexas.

Um sistema de base de dados espacial pode ser definido da seguinte forma (Güting, 1994):

Um sistema de base de dados espacial é um sistema de base de dados de pleno direito;

Disponibiliza tipos de dados espaciais tanto no seu modelo de dados como na linguagem de pesquisa ("*query language*");

Utiliza tipos de dados espaciais na sua implementação, assegurando, no mínimo, métodos para criação de índices espaciais e operações de associação ("*join*") espaciais.

O autor referido justifica os requisitos enunciados acima da forma que se descreve nos seguntes parágrafos.

O primeiro requerimento pretende enfatizar o facto de que a informação espacial ou geométrica está, na prática, sempre associada a informação "não espacial", por exemplo a

dados alfa-numéricos. Um sistema que não possibilite a modelação de dados comuns bem como a execução de pesquisas por atributos alfa-numéricos não terá grande utilidade prática para os seus utilizadores. Desta forma, o autor considera que um sistema de bases de dados espaciais (SBDE) corresponderá sempre a um sistema de bases de dados (SBD) de pleno direito, com funcionalidades adicionais para tratamento de dados espaciais.

Tipos de dados espaciais, como Ponto, Linha, Região, fornecem as abstracções necessárias à modelação de entidades geométricas no espaço, bem como à modelação das relações entre essas entidades (L intersecta R), das suas propriedades (área(R) > 1000) e operações (intersecção(L, R) -> parte de L contida em R). Os tipos de dados utilizados irão variar, dependendo do tipo de aplicações que se pretende implementar (rectângulos em desenho de circuitos electrónicos, superfícies e volumes em aplicações de modelação 3D). Sem tipos de dados espaciais, um SBD não oeferece um suporte adequado à modelação.

Quanto ao terceiro requerimento, um SBDE necessita no mínimo de ser capaz de identificar, num extenso conjunto de objectos existentes num dado espaço, quais os que se localizam no interior de uma área particular sem necessitar de pesquisar todo o conjunto. Ou seja, a possibilidade de implementar índices espaciais é indispensável num SBDE. Um SBDE deverá também assegurar associações ("*joins*") entre objectos de diferentes classes, através de relações espaciais, de forma mais eficiente que por simples filtragem do produto cartesiano das classes em questão.

2.2. Evolução dos SBDE

A empresa Boundless Spatial, Inc., apresenta no seu sítio Web, mais precisamente na página http://workshops.boundlessgeo.com/postgis-intro/, uma excelente síntese da evolução dos SBDE. Assim, são descritas três etapas na evolução destes sistemas:

Primeira geração – implementação de Sistemas de Informação Geográfica (SIG) em que os todos os dados espaciais são armazenados em ficheiros simples, não indexados e sem relações estruturadas entre si ("*flat files*"). Necessitam de aplicações SIG para interpretação e manipulação de dados. São concebidos para dar resposta às necessidades dos utilizadores de informação totalmente contida dentro de um domínio organizacional específico, utilizando sistemas proprietários para lidar com informação espacial.

- Segunda geração –SIG que utilizam bases de dados relacionais para armazenamento de parte dos dados, normalmente atributos não-espaciais.

- Terceira geração – bases de dados espaciais "propriamente ditas", originadas a partir do momento em que se começou a considerar entidades espaciais ("*features*") como objectos primários de bases de dados, de "pleno direito" ("*first class database objects*"). As bases de dados espaciais apresentam uma integração total da informação espacial numa base de dados relacional, reflectindo uma mudança da filosofia centrada em SIG, característica das gerações anteriores.

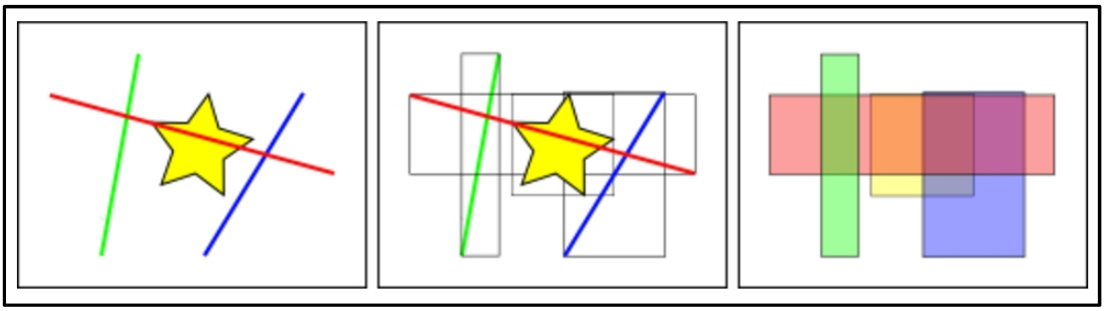
Refira-se ainda que actualmente é frequente encontrarem-se exemplos de análises de dados com uma componente temporal, quando é possível associar a conjuntos de dados espaciais ("*datasets*") informação do tipo data-hora. São exemplos deste tipo de utilizações o estudo de frequências de trajectos em transportes públicos, ou análises de evolução morfológica em ambientes dinâmicos (Heyer, 2018).

2.3. Índices espaciais

Os sistemas de bases de dados relacionais, para dados não espaciais, utilizam índices constituidos por estruturas em árvore, hierarquizadas, para acelerar consultas e acesso aos dados. Os índices espaciais apresentam uma aproximação diferente, em que não são indexados directamente os diferentes objectos geométricos mas sim as envolventes ("*bounding boxes*") destes objectos.

Na figura 2.1, podemos ver que apenas a linha vermelha intersecta a estrela amarela. No entanto, quando se utilizam as envolventes das diferentes entidades (linhas e estrela) verifica-se que duas envolventes (das linhas azul e vermelha) intersectam a envolvente da estrela.

Figura 2.1 -*Exemplo de índices espaciais*



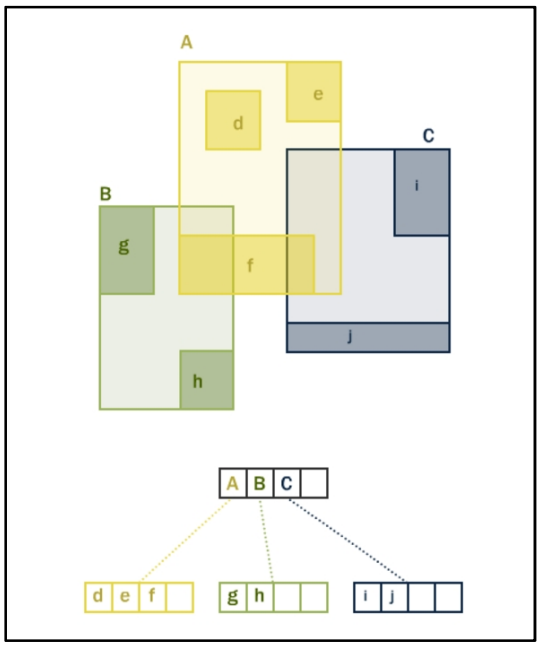
Nota. Adaptado de http://workshops.boundlessgeo.com/postgis-intro/.

Para responder à questão de quais as linhas que intersectam a estrela amarela, uma base de dados que utilize este tipo de índices espaciais irá em primeiro lugar determinar quais as "*bounding boxes*" que interceptam a envolvente da estrela, utilizando o índice espacial, que permite uma resposta muito rápida, e, num segundo passo, proceder a um cálculo exacto de quais as linhas que intersectam a estrela, considerando apenas as entidades identificadas no primeiro teste.

Para tabelas com um elevado número de entidades geométricas, este tipo de aproximação, em duas etapas, utilizando primeiro índices espaciais para uma resposta aproximada e proceder a uma determinação exacta apenas para o subconjunto de entidades devolvidas no primeiro passo, permite reduzir drasticamente o número de cálculos necessários à execução de pesquisas.

Tanto o SBDE como PostGIS e Oracle Spatial utilizam estruturas de índices espaciais do tipo "R-tree". Estas estruturas agrupam entidades em rectângulos, subdividos sucessivamente. Apresentam capacidade de auto-geração, conseguindo indexar objectos com diferentes densidades e tamanhos de forma automática (Figura 2.2).

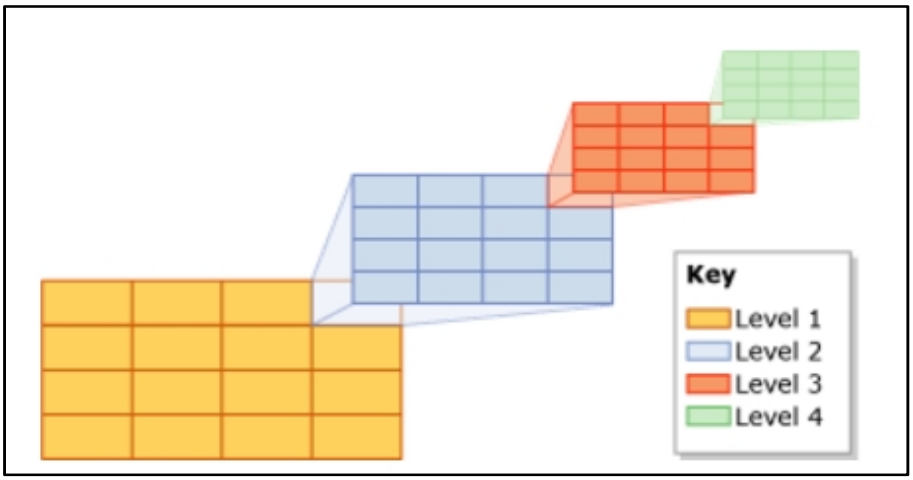
Figura 2.2 - *Ilustração de hierarquia de índices espaciais R-tree*



Nota. Adaptado de http://workshops.boundlessgeo.com/postgis-intro/.

No caso do SBDE da Microsoft, SQL Server, é utilizado outro tipo de estrutura de índices espaciais. De acordo com a descrição presente na página Web https://docs.microsoft.com/en-us/sql/relational-databases/spatial/, os índices espaciais são construídos com recurso a estruturas do tipo B-tree. O processo de geração de índices passa por uma decomposição hierárquica do espaço, em 4 níveis, referenciados de 1 a 4, sendo o nível 1 o superior. Em cada nível, todas as "*grids*" possuem o mesmo número de células ao longo dos dois eixos (4x4, 8x8) e as células apresentam todas o mesmo tamanho (Figura 2.3).

Figura 2.3 – *Ilustração de hierarquia de "grid"*

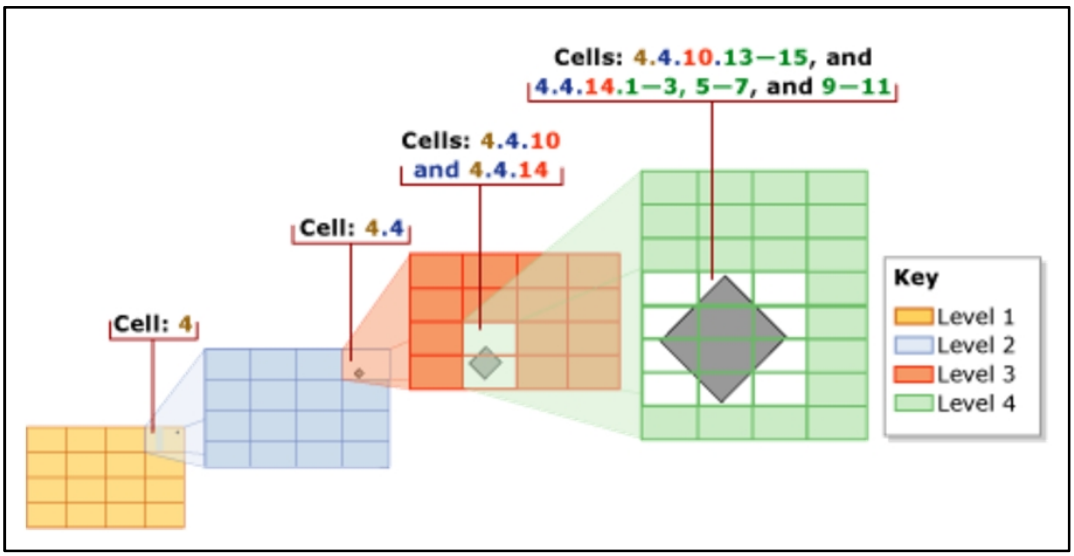


Nota. Adaptado de https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-indexes-overview.

As células de cada nível são numeradas de forma linear, utilizando uma variação de curva de Hilbert. Na imagem da figura 2.4, é utilizada no entanto, uma numeração sequencial simples, ao longo de linhas sucessivas, para uma explicação simplificada do processo. Os diferentes polígonos e linhas representados (que poderiam corresponder a edifícios e ruas, por exemplo) estão coberto por uma grelha 4x4, de nível 1. As células deste nível são numeradas de 1 a 16, iniciando-se a contagem no topo esquerdo.

Após a decomposição de um espaço numa hierarquia de grelhas, o índice espacial é construído para cada objecto, através de um processo de composição. O objecto em causa é associado a um conjunto de células "tocadas" em cada nível, começando no nível 1 da hierarquia, até percorrer todos os restantes níveis. O resultado final deste processo é uma lista de células "tocadas" que é registada no índice espacial. Desta forma é possível localizar um determinado objecto no espaço, relativamente a outros objectos da mesma tabela (e coluna) indexados da mesma forma.

Figura 2.4 – *Ilustração do processo de indexação*



Nota. Adaptado de https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-indexes-overview.

2.4. Relações espaciais

Uma das funções básicas de um SBDE é a determinação das relações espaciais entre entidades. A determinação de distâncias entre um aterro sanitário e zonas residenciais constitui um exemplo de uma relação espacial. No caso em estudo, será importante assegurar a execução de pesquisas com base em critérios espaciais para, por exemplo, localizar dados existentes numa determinada área geográfica. Os SBDE existentes apresentam funcionalidades intrínsecas para estes fins, permitindo averiguar se uma geometria apresenta interseção válida com outra, ou se está totalmente contida na segunda (Corti, 2104, pág.112-116). Desta forma, é minimizado, de forma o esforço de programação para se desenvolverem funcionalidades deste tipo.

2.5. Principais SBDE existentes

.Actualmente, podemos considerar três SBDE principais: Oracle Spatial, SQL Server e PostgreSQL com extensões para dados espaciais PostGIS. Os três sistemas apresentam um conjunto completo de tipos e funcionalidades espaciais, mas apenas o SBDE PostgreSQL /PostGIS se apresenta como software livre, sem a imposição de licenças comerciais.

2.6. Escolha do sistema de base de dados espaciais a utilizar no projeto

Face ao pressuposto de utilização de software livre, e dada a rica funcionalidade apresentada, foi escolhido o sistema de bases de dados PostgreSQL (The PostgreSQL Global Development Group, 2014), complementado com o conjunto de extensões para dados espaciais denominado PostGIS, que assegura todas as funcionalidades de um moderno SBDE. Este software possui licença do tipo GPL.

Para além de incluir índices e associações espaciais, como já descrito acima, apresenta uma implementação completa de tipos de entidades espaciais, nomeadamente pontos, linhas poligonais ("*linestrings*"), polígonos, multi-pontos, multi-poligonais, multi-polígonos e colecções de geometrias ("*geometrycollection*").

Permite ainda a execução de pesquisas por critérios espaciais utilizando a linguagem SQL, e implementa os padrões definidos pelo Open Geospatial Consortium (OGC) para "*Simple Features for SQL*". Para gestão das bases de dados, criação de tabelas e "*scripts*" foi utilizada a aplicação PgAdmin 3, versão 1.18.

2.7. Estruturas de dados tipo "*Pointcloud*"

Com a rápida evolução de sistemas de aquisição de dados espaciais de que resulta elevada densidade de informação, como os sistemas multifeixe no mar ou LIDAR para levantamentos terrestres, tornou-se necessário desenvolver novas formas de armazenamento para fazer face à enorme quantidade de informação. Foi neste quadro que Ramsey (2012), propôs a adoção de estruturas do tipo "*pointcloud*" e "*pcpatch*". Este autor descreve que grande parte da complexidade de trabalhar com dados provenientes de sistemas LIDAR se prende com o o facto de poderem existir inúmeras variáveis para cada ponto adquirido. Estas dependem do tipo de sistema utilizado e do processo de aquisição. Alguns sistemas apenas apresentam informação X,Y,Z (coordenadas planimétricas e cotas) enquanto que outros apresentam valores de intensidade do sinal refletido, número de reflexões, valores de cor (RGB), entre muitas outras variáveis possíveis.

O autor assinala ainda que também não existe consistência na forma de armazenar as diferentes informações, valores de intensidade podem ser gravados com inteiros de 4 bytes ou de apenas um byte, os valores de X,Y e Z podem ser representados como reais de dupla precisão ou como inteiros com factor de escala associado.

As estruturas "pointcloud" propostas por este autor resolvem este problema com a utilização de um documento esquema onde são definidos os diferentes dados e a sua forma de armazenamento. O documento esquema proposto para utilização em PostgreSQL (PostGIS) é compatível com a livraria PDAL (*Point Data Abstraction Layer*). A utilização deste tipo de documento permite a definição, para um dado conjunto de dados LIDAR, de quais as variáveis presentes, seus tipos e formas de armazenamento (com e sem escala).

O tipo base para estruturas "*pointcloud*" é o "*PcPoint*". Cada "*PcPoint*" pode ter um número variável de atributos, mas no mínimo possui coordenadas X e Y para localização no espaço.

As entidades do tipo "*PcPoint*" são agrupadas em "*PcPatches*", por critérios de proximidade. Desta forma, é possível armazenar numa base de dados um número

relativamente reduzido de entidades quando comparado com o número de pontos individuais que seria necessário considerar sem este tipo de estruturas.

O documento esquema descritivo da constituição da estrutura "*PcPoint*" utilizada é gravado na base de dados, em tabela própria, com um código identificador único. Este código é utilizado para a construção dos "*PcPatches*" a partir dos "*PcPoints*" individuais. Quando se necessitar aceder à informação "encapsulada" nestas estruturas é através deste código que é descodificada, interpretando o documento esquema correspondente.

3. METODOLOGIA

3.1. Estrutura de dados – concepção e implementação

3.1.1. Definição de esquema para estrutura "*Pointcloud*"

Para implementação das funcionalidades espaciais no sistema PostgreSQL, foi necessário instalar as extensões "PostGIS", "pointcloud" e "pointcloud_PostGIS", através de instruções executadas sobre uma base de dados de teste:

"*CREATE EXTENSION PostGIS;*
CREATE EXTENSION pointcloud;
CREATE EXTENSION pointcloud_PostGIS;"

Após estas operações, são disponibilizadas pelo SBDE, de forma automática, as seguintes tabelas e "*views*":

- "*spatial_ref_sys*", tabela com mais de 3000 sistemas de referência espaciais e respectivas fórmulas de conversão/reprojecção entre eles, armazenadas na coluna "proj4text" . O SBDE PostGIS utiliza a livraria Proj4 para transformação de coordenadas e reprojecções;

- "*geometry_columns*", "*view*" que identifica todas as colunas (campos) correspondentes aos diferentes tipos de geometrias presentes na base de dados bem como as tabelas onde ocorrem;

- "*poincloud_formats*", tabela com os formatos definidos pelo utilizador para as estruturas de dados do tipo PCpoint ("pointcloud point"), expressos em linguagem XML;

- "*pointcloud_columns*", "*view*" que identifica todas as colunas correspondentes a tipos "*pointcloud point*" presentes na base de dados bem como as tabelas onde ocorrem;

Como descrito no capítulo anterior, foi previsto para este projeto a utilização de estruturas do tipo "*Pointcoud*", agrupadas em blocos do tipo "*PCPatch*", para a integração dos dados espaciais primários (batimétricos e altimétricos) . Para assegurar um eficiente armazenamento dos dados procedeu-se a uma escolha criteriosa dos atributos essenciais a preservar. Foram definidos os seguintes:

- coordenadas x e y expressas em latitude e longitude WGS84 (graus), armazenados sob a forma de inteiros de 4 bytes ("int32_t"), com factor de escala de 10^{-7}, representando uma amplitude de valores possíveis entre -214.7483648 a 214.7483647, permitindo a representação da gama completa de valores para a latitude (+90º a -90º) e longitude (-180º a +180º) e resolução equivalente a 11.17 milimetros (projecção UTM no plano cartográfico) para a situação menos favorável (latitude no equador);

- profundidade/altitude (m), referida ao nivel médio das águas do mar, por defeito, armazenada como inteiro de 4 bytes, com factor de escala de 10^{-4}, permitindo uma resolução de uma décima de milímetro e cobrindo largamente a amplitude de profundidades e altitudes possíveis no planeta (gama de valores: -214748.3648 a 214748.3647);

- valor de intensidade de retorno /retrodispersão acústica, armazenado sob a forma de inteiro de 2 bytes (int16_t), com factor de escala de 10^{-2} (gama de valores: -327.68 a 327.67), considerado adequado para representar os valores destes parâmetros identificados na bibliografia existente (normalmente entre -10 a -60 dB para retrodispersão acústica, enquanto os ficheiros LAS utilizam inteiros de 2 bytes para codificação da intensidade de retorno do sinal laser) ;

- de forma a se poder incluir informação de cor (no caso de informação LIDAR processada com fotografia aérea) ou possibilitar o armazenamento de informação adicional (classificações de tipos de sedimento, rocha, litologias) foram ainda considerados três campos adicionais, designados por R,G,B, codificados com 1 byte cada, sem escala.

A estrutura descrita acima foi adaptada para esquema XML (compativel com a livraria *PDAL - Point Data Abstraction Library*) e inserida na tabela "*pointcloud_formats*", como

um campo de texto, juntamente com um ”*id*” numérico e uma chave externa para a tabela “*spatial_ref_sys*” para definição do sistema de coordenadas associado. No parágrafo seguinte apresenta-se parte inicial do esquema XML, correspondente à definição da coordenada “x”.

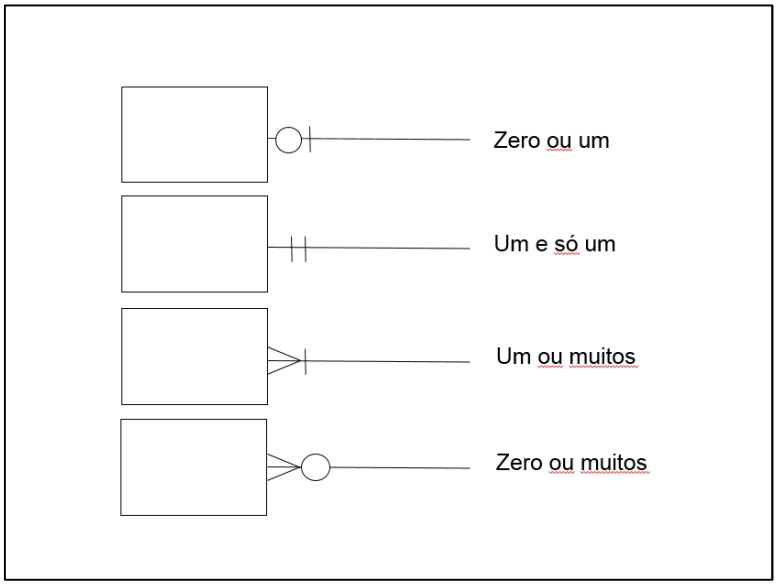
“INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (3, 4326, '<?xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <pc:dimension>
  <pc:position>1</pc:position>
  <pc:size>4</pc:size>
  <pc:description>X : Longitude wgs84 como inteiro de 4 bytes, com escala de 10^(-7)</pc:description>
  <pc:name>X</pc:name>
  <pc:interpretation>int32_t</pc:interpretation>
  <pc:scale>0.0000001</pc:scale>
 </pc:dimension>
 <pc:dimension> ”

3.1.2. Principais entidades e relações

De forma a melhor ilustrar as diferentes entidades e relações entre elas, adoptou-se neste trabalho a notação do tipo “*Crow's Foot*” , em que as entidades são representadas por “caixas” rectangulares e as relações por linhas. A cardinalidade das relações é representada por diferentes simbologias, consoante as relações são do tipo um para muitos, um para um, muitos para muitos.

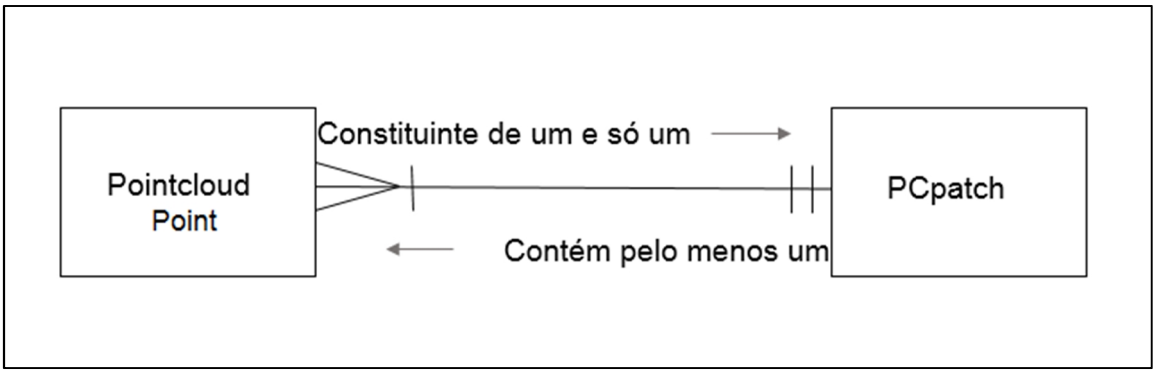
Na figura abaixo são mostrados exemplos das simbologias utilizadas para descrição das relações entre entidades (Figura 3.1).

Figura 3.1 – *Notação "Crow's Foot" para relações entre entidades*



Estabelecida a estrutura "*Pointcloud*" a utilizar, foram definidas as tabelas correspondentes à entidades primárias. Seguindo a filosofia já apresentada no capítulo anterior, pretende-se integrar dados batimétricos e altimétricos, organizados por campanhas de aquisição (cruzeiros, levantamentos, voos) numa tabela de entidades "*pc_points*" (abreviatura de "*Pointcloud points*") , num primeiro passo. O conjunto de entidades "*pc_points*" é em seguida, agrupado em blocos do tipo "*PCpatch*", constituindo estes objectos o tipo primário armazenado na base de dados. Após esta operação, todos as entidades da tabela "*pc_points*" são apagadas, de forma a se poder repetir o processo para o próximo conjunto de dados a integrar (Figura 3.3).

Figura 3.2 – *Entidades principais para registo de dados batimétricos e altimétricos*



Nota.Cada entidade "*PCpatch* "integra pelo menos uma entidade "*Pointcloud point"*.

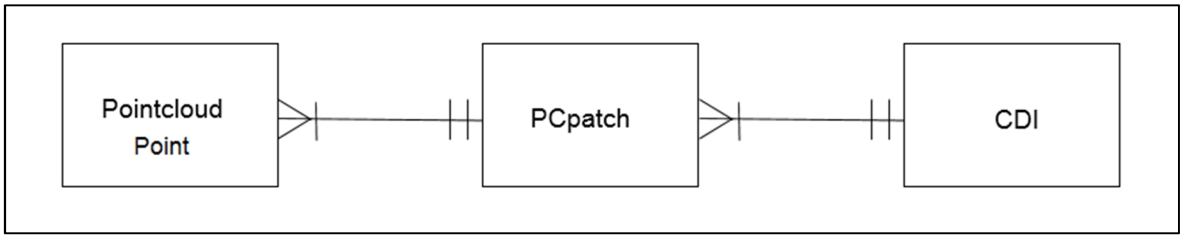
O número máximo de entidades "*Pointcloud point*" a agrupar em cada bloco "*PCpatch*" é determinado pelo espaço de armazenamento necessário à estrutura de dados definida pelo respectivo esquema XML. Caso se ultrapasse o limite de 2 KB para cada linha ("*row*") da tabela, o sistema PostgreSQL irá criar, de forma automática, uma tabela suplementar do tipo "*TOAST*" (*The Oversized-Attribute Storage Technique*) para armazenar estes registos, em fragmentos de menor dimensão, registando na tabela original um ponteiro para acesso a esta informação, em vez dos dados originais.

Dado que existe um limite para o tamanho de tabelas do tipo "*TOAST*" (na ordem de 2^{32} registos) e de forma a manter a estrutura de tabelas o mais simples possível, prevendo necessidades futuras de reorganização da base de dados, e ainda de forma a garantir o acesso directo, mais rápido, aos dados, pretende-se evitar a necessidade de se gerarem tabelas deste tipo. Para isso será necessário manter o número máximo de "*pc_points*" num bloco "*PCpatch*" abaixo de um determinado limite de forma a não se ultrapassar o limite de 2 KB para cada bloco. Após alguns ensaios determinou-se que é possível registar até 600 pontos por bloco, sem que o sistema necessite de criar o mecanismo referido. Será este o número máximo de pontos a considerar nas operações de agregação em blocos "*PCpatch*".

Embora os dados primários sejam armazenados sob a forma de "*PCpatches*", estes estão hierarquizados de acordo com o cruzeiro ou levantamento onde foram adquiridos. Dado que o projeto de tese teve por base a integração com estruturas de dados do projeto Emodnet, em que é utilizado um formato específico de metadados ("*Common Data Index*") para cada conjunto de dados correspondente a uma campanha de aquisição, adoptou-se a designação "CDI" para descrever esta entidade.

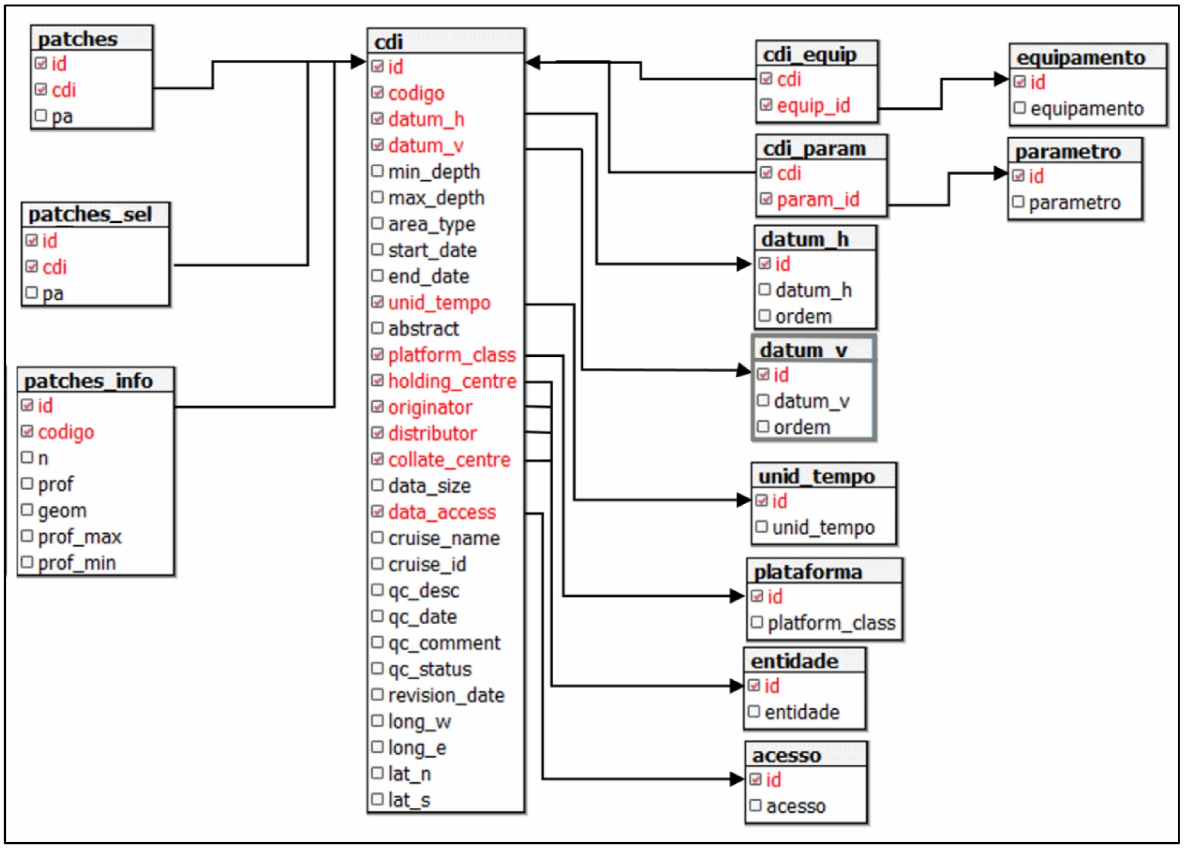
Desta forma, o esquema completo de entidades primárias necessárias para a correcta integração dos dados integra as entidades "*Pointcloud point*" (ponto), "*PCpatch* "(conjunto de pontos) e CDI (Figura 3.4).

Figura 3.3 – *Diagrama de entidades principais para registo de dados e metadados*



Para integração de toda a informação necessária na base de dados foi necessário desenvolver conjunto de tabelas e relações para as quais se apresenta um diagrama na Figura 3.4.

Figura 3.4 – *Diagrama com as principais tabelas e relações*



Nota. Os campos correspondentes a chaves primárias e chaves externas estão assinalados a vermelho.

3.1.2.1. Tabela Patches

Foi definida no SBDE a tabela Patches para armazenamento das entidades "*PCpatch*", Esta tabela apresenta os seguintes campos:

id – chave primária, código numerico sequencial, gerado automáticamente pelo SBDE ;

cdi – chave externa para tabela "CDI", onde são registados os metadados de cada conjunto de dados (correspondentes a cruzeiro, levantamento);

pa – estrutura "*PCpatch*", definida pela extensão espacial "*pointcloud*", integrando conjunto de estruturas do tipo "*pointcloud point*".

3.1.2.2. Tabela Patches_info

Esta tabela foi criada de forma a sintetizar informação de conjuntos de "*PCpatches*" correspondentes aos vários "*datasets*". Para cada conjunto de dados importado para a base de dados é criado um registo nesta tabela, que inclui um campo geométrico (poligono) com o contorno da área correspondente. Esta tabela inclui os seguintes campos:

id – Identificador numerico (chave externa para tabela CDI) do conjunto de dados;

codigo – Identificador único, para o projeto Emodnet, do cruzeiro que originou o conjunto de dados (chave externa para tabela CDI);

n – Número total de pontos com informação de batimetria (campo calculado a partir do somatório de pontos em cada um dos "*patches*" integrantes do conjunto de dados);

prof – Profundidade média do "*dataset*" (campo calculado a partir dos valores médios de profundidade para cada "*patch*");

prof_max – Profundidade máxima do "*dataset*" (campo calculado a partir dos valores máximos de profundidade para cada "*patch*");

prof_min – Profundidade miníma do "*dataset*" (campo calculado a partir dos valores minímos de profundidade para cada "*patch*");

geom – Campo geométrico (polígono) correspondente à envolvente exterior do conjunto de "*patches*" integrantes do "*dataset*".

3.1.2.3. Tabela Sel_Patches

Trata-se de uma tabela temporária, onde é armazenado o resultado de operações de pesquisa realizadas pelo utilizador do programa de exploração de dados. É utilizada exclusivamente para visualização. Inclui os mesmos campos da tabela Patches:

id – Chave primária ;

cdi – Chave externa para tabela "CDI";

pa – Estrutura "*PCpatch*".

3.1.2.4. Tabela CDI

Para armazenamento dos metadados associados a cada "*dataset*", normalmente correspondente a um cruzeiro, campanha ou levantamento específico, foi criada a tabela CDI. É esta tabela que centraliza toda a informação de metadados necessária à identificação do conjunto de dados e à sua divulgação/partilha no quadro do projeto Emodnet.

Note-se que, devido à necessidade de se incluírem atributos como equipamentos utilizados e parâmetros medidos (entre outros), que podem resultar em multiplas entradas para um mesmo registo, foi necessário criar tabelas de "ligação" entre uma entidade CDI (correspondente a um registo único) e tabelas auxiliares com listas de parâmetros, equipamentos e outros. Para visualizar toda a informação associada a cada entidade CDI será necessário proceder a consultas ("*queries*") do tipo "*JOIN*" integrando a tabela principal, as tabelas de ligação e as tabelas secundárias.

A tabela CDI apresenta os seguintes campos:

id - Chave primária, código numérico sequencial, gerado automáticamente pelo SBDE;

codigo – Identificador único, para o projeto Emodnet, do cruzeiro que originou o conjunto de dados. Campo de texto. Corresponde ao atributo "Dataset-id" no formato XML utilizado para partilha de dados no âmbito da estrutura Sea Data Net;

cruise_name – Nome do cruzeiro;

cruise_id – Código do cruzeiro;

datum_h – Chave externa para tabela Datum_H, identificando o datum horizontal original utilizado no posicionamento da campanha/levantamento em que os dados foram adquiridos. Note-se que em toda a base de dados são utilizadas exclusivamente coordenadas geográficas no sistema WGS84. No processo de importação de dados é feita a conversão para este sistema a partir do datum original.

datum_v - Chave externa para tabela Datum_V, identificando o datum vertical original utilizado na campanha/levantamento em que os dados foram adquiridos. Preferencialmente, a informação de batimetria deverá estar referida ao Zero Hidrográfico, ou equivalente (LAT), para zonas costeiras, e ao nível médio do mar para levantamentos oceânicos .

long_w – Valor de longitude (WGS84) para o extremo oeste da área correspondente ao conjunto de dados ("*bounding box*");

long_e - Valor de longitude para o extremo este da área correspondente ao conjunto de dados;

lat_n - Valor de latitude para o extremo norte da área correspondente ao conjunto de dados;

lat_s - Valor de latitude para o extremo sul da área correspondente ao conjunto de dados;

min_depth – Valor da profundidade mínima do conjunto de dados, em metros. Os dados de profundidade são armazenados na base de dados como valores negativos, utilizando o tipo "*real – single precision*";

max_depth – Valor da profundidade máxima do conjunto de dados, em metros.;

start_date – Data e hora do início do cruzeiro;

end_date – Data e hora do final do cruzeiro;

abstract – Resumo das actividades levadas a cabo no cruzeiro/levantamento;

holding_centre – Código SeaDataNet da instituição detentora dos dados

originator - Código SeaDataNet da instituição que procedeu à recolha dos dados (chave externa para tabela Entidades);

distributor - Código SeaDataNet da instituição distribuidora dos dados (chave externa para tabela Entidades);

collate_centre - Código SeaDataNet da instituição organizadora dos dados (chave externa para tabela Entidades);

qc_description – Descrição da metodologia de controlo de qualidade seguida na aquisição de dados;

qc_comment – Comentários adicionais ao controlo de qualidade;

qc_date – Data do controlo de qualidade;

qc_status – Estado do controlo de qualidade (Verdadeiro/Falso);

data_size – Tamanho, em Megabytes, do ficheiro (Ascii, xyz) correspondente ao conjunto de dados;

data_access – Tipo de acessibilidade dos dados (livre, por negociação), chave externa para tabela Acesso;

revision_date – Data da última alteração/revisão dos metadados;

3.1.3. Tabelas auxiliares

De forma a cumprir com os objectivos do projeto, nomeadamente o armazenamento de metadados relativos à informação primária, foram criadas tabelas auxiliares para os diferentes tipos de atributos. Estas cumprem com as especificações definidas pelas infra-estruturas pan-europeias de divulgação e partilha de dados com que o IPMA colabora, nomeadamente o projeto Emodnet Bathymetry.

Os dados para estas tabelas foram importados a partir de ficheiros XML disponibilizados no âmbito da infraestrutura SeaDataNet da União Europeia, e correspondem a vocabulários mantidos pela organização British Oceanographic Data Centre (BODC).

3.1.3.1. Entidade

Tabela com dados de identificação de instituições e empresas com responsabilidades na aquisição de dados batimétricos e/ou na sua custódia e distribuição. Numa primeira fase, incluíu-se apenas a designação oficial da instituição e um código de identificação ("*id*") numérico.

3.1.3.2. Datum_H

Tabela com lista de sistemas de coordenadas utilizados na aquisição de dados (oceanográficos, batimetria, LIDAR). Inclui a designação do sistema e um código de identificação ("*id*") numérico idêntico ao código EPSG (European Petroleum Survey Group).

3.1.3.3. Datum_V

Tabela com lista de referenciais verticais utilizados na aquisição de dados. Inclui a designação do referencial (datum vertical) e um código de identificação ("*id*") .

3.1.3.4. Equipamento

Tabela com descrições de equipamentos que tenham sido utilizados para aquisição de dados. Inclui a designação/descrição do equipamento e um código de identificação ("*id*") numérico.

3.1.3.5. Parâmetro

Tabela com descrições de parâmetros medidos durante as campanhas de mar. Inclui a designação do parâmetro e um código de identificação ("*id*").

3.1.3.6. Plataforma

Tabela com diferentes tipos de plataforma utilizada na aquisição de dados (navio, pequena embarcação, helicóptero e outros). Inclui a designação da plataforma e um código de identificação ("*id*").

3.1.4. Tabelas de ligação

3.1.4.1. Cdi_equip

Tabela que associa conjuntos de equipamentos aos diferentes registos da tabela CDI. De forma a materializar a relação de "muitos para muitos" entre os registos CDI (cruzeiros, levantamentos) e os diferentes equipamentos utilizados foi implementada esta tabela. Os campos constituintes resumem-se a chaves externas para as tabelas CDI e Equipamento. Para cada equipamento utilizado num cruzeiro é criado um registo com o código identificador do cruzeiro (chave externa para a tabela CDI) e com o código identificador do equipamento (chave externa para a tabela Equipamento).

3.1.4.2. Cdi_param

Tabela que associa conjuntos de parametros medidos aos diferentes registos da tabela CDI. Análoga à tabela anterior , para materializar a relação de "muitos para muitos" entre os registos CDI (cruzeiros, levantamentos) e os diferentes parâmetros medidos. Os campos constituintes resumem-se a chaves externas para as tabelas CDI e Parametro.

3.1.5. "*Views*" e funções

Para assegurar maior eficiência na exploração dos dados recorreu-se à elaboração de "*views*" e funções, utilizando linguagem SQL. Uma "*view*" consiste numa pesquisa ("*query*"), previamente gravada e testada na própria base de dados. As "*views*" são especialmente úteis para simplificar a consulta de informação de uma entidade que se encontra armazenada em várias tabelas relacionadas por um identificador ("*id*") comum. O SBDE PostGIS possibilita ainda a definição de funções que permitem a extracção de informação quantitativa ou geométrica a partir de operações sobre os dados das diferentes tabelas.

Nos pontos seguintes descrevem-se as "*views*" e funções desenvolvidas.

3.1.5.1. "*View*" – vv_patches

Analisando a sintaxe da expressão do parágrafo seguinte, identificam-se as funções "*geometry"*, "*st_centroid*", "*pc_patchavg"* e "*pc_numpoints*".

"*SELECT patches.cdi, patches.id, patches_info.codigo, geometry(patches.pa) AS geom, st_centroid(geometry(patches.pa)) AS pp, pc_patchavg(patches.pa, 'z'::text) AS prof, pc_numpoints(patches.pa) AS n FROM patches LEFT JOIN patches_info ON patches.cdi = patches_info.id;*"

As duas primeiras funções são integrais à extensão PostGIS do sistema de base de dados PostgreSQL, devolvendo uma geometria e um ponto x,y, respetivamente. As duas últimas fazem parte da extensão Pointcloud, e devolvem o valor médio do parâmetro numérico (indicado dentro de parênteses) para cada "*patch*" e o número total de pontos (tipo "*pointcloud*") presentes em cada "*patch*". A "*view*" mostrada acima agrega dados da tabela Patches e Patches_info, procedendo a uma "junção" ("*LEFT JOIN*") das duas tabelas , relacionadas entre si pelo campo "cdi" da primeira tabela e "id" da segunda (embora com designações diferentes ambos se referem à chave primária de cada conjunto de dados).

Na prática, esta tabela virtual nunca é preenchida com a totalidade dos registos, quando o programa de gestão de dados executa uma consulta com esta "*view*", é sempre restringida a um único conjunto de dados (através do respectivo "*id*").

3.1.5.2. "*View*" – vv_patches_sel

A seguinte "*view*" foi elaborada para permitir a visualização de "*patches*" identificados através de pesquisa espacial ("*bounding box*") e armazenados na tabela temporária "Patches_sel" .

"*SELECT patches_sel.cdi, patches_sel.id, cdi.codigo, geometry(patches_sel.pa) AS geom, st_centroid(geometry(patches_sel.pa)) AS pp, pc_patchavg(patches_sel.pa, 'z'::text) AS prof, pc_numpoints(patches_sel.pa) AS n FROM patches_sel LEFT JOIN cdi ON patches_sel.cdi = cdi.id*;"

Devolve os campos "cdi" e "id" da tabela "patches_sel" e o campo "codigo" da tabela "Cdi". À semelhança da anterior, utiliza o operador SQL "*LEFT JOIN*" para ligar estas duas tabelas através de campo em comum. Notar a utilização da função PostGIS "*st_centroid*" para obter o ponto central de cada "*patch*", da função Pointcloud "*pc_patchavg*" para extrair o valor médio das profundidades e da função "*pc_numpoints*" para determinar o número total de pontos.

3.1.5.3. Função – f_pontos

A função abaixo representada foi criada para consultar a informação de batimetria contida numa estrutura "*Pcpatch*".

"CREATE OR REPLACE FUNCTION public.f_pontos(IN parm1 integer, IN parm2 integer) RETURNS TABLE(n bigint, id integer, cdi integer, pt geometry, z numeric) AS

$BODY$

SELECT row_number() over() as n, temp.id, temp.cdi, st_makepoint(temp.x::double precision, temp.y::double precision) AS geom, temp.z AS prof FROM (SELECT patches.id, patches.cdi,pc_get(pc_explode(patches.pa),'x'::text) AS x,

pc_get(pc_explode(patches.pa) , 'y'::text) AS y,

pc_get(pc_explode(patches.pa), 'z'::text) AS z FROM patches WHERE patches.cdi = $1 And id = $2) temp;

$BODY$"

Analisando a função a partir da expressão SELECT mais interior, contida entre parênteses e a primeira a ser efectuada, vemos como são utilizadas as funções "*Pointcloud*" "*pc_get*" e "*pc_explode*" para extrair as coordenadas x e y, bem como a profundidade (z) de um determinado "*patch*" com códigos "cdi" (identificação do "*dataset*") e "*id*" (identificação do "*patch*") dados pelos argumentos da função ($1 e $2). Na expressão SELECT exterior os valores de x e y são passados à função PostGIS "*st_makepoint*" para construir uma entidade geométrica (do tipo "*point*" 2D) para cada par. A função "f_pontos" devolve um conjunto de registos com entidades do tipo "*point*" e respectivos valores de profundidade, bem como um nº de ordem (n) para cada. Esta função é utilizada pelo programa de gestão de dados para visualização, em mapa, dos pontos constituintes de um único "*patch*" escolhido pelo utilizador.

3.1.5.4. Função – f_pontoscdi

A função "f_pontoscdi" foi criada para facilitar a visualização em mapa de todos os pontos constituintes de um conjunto de dados (incluindo todos os "*patches*").

"CREATE OR REPLACE FUNCTION public.f_pontoscdi(IN parm1 integer)

RETURNS TABLE(n bigint, id integer, cdi integer, pt geometry, z numeric) AS

$BODY$

SELECT row_number() over() as n, temp.id, temp.cdi,
st_SetSrid(st_makepoint(temp.x::double precision, temp.y::double precision), 4326)
AS geom, temp.z AS prof FROM (SELECT patches.id, patches.cdi,
pc_get(pc_explode(patches.pa), 'x'::text) AS x, pc_get(pc_explode(patches.pa),
'y'::text) AS y, pc_get(pc_explode(patches.pa), 'z'::text) AS z FROM patches
WHERE patches.cdi = $1) temp;
$BODY$"

É muito semelhante à anterior, mas apenas necessita de um argumento, o identificador único do "*dataset*" ("patches.cdi"). De forma a poder ser utilizada também para exportação dos dados para o formato "*shapefile*" (com a informação do sistema de coordenadas) , foi utilizada a função PostGIS "*st_SetSrid*" para atribuir este metadado ao resultado da função "*st_makepoint*" (passando o código numérico correspondente ao sistema WGS84, 4326). Desta forma, assegura-se a correcta identificação da geometria (pontos) para utilização noutro software (como ArcMap, por exemplo).

3.2. Desenvolvimento do programa de gestão de dados

Foi definido como um dos objectivos do presente trabalho o desenvolvimento de aplicação informática que permita executar funções de gestão da base de dados descrita nos pontos anteriores, nomeadamente a criação de novas bases de dados, importação de dados, preenchimento de atributos e exportação de dados e metadados. Considerou-se ainda a criação de ferramentas para visualização e pesquisa.

Pretende-se, nas páginas seguintes, expor as técnicas e metodologias utilizadas.

3.2.1. Linguagem de programação e livrarias

Foi escolhida como linguagem de programação para execução do programa a linguagem Python na sua versão 2.7. Esta escolha baseou-se , essencialmente em três fatores principais:

- Alguma experiência prévia na utilização da linguagem Python, especialmente durante a componente curricular do curso de mestrado;
- Acessibilidade da linguagem Python;
- Disponibilidade de livrarias apropriadas ao objectivo (integração com PostGIS e QGis, interfaces gráficas);

Para além disto, também foi fator determinante a possibilidade de utilizar a linguagem Python, bem com as todas as livrarias necessárias, tanto em ambiente Windows como Linux.

Para comunicação e interacção com a base de dados utilizou-se a livraria Psycopg (versão 2.4). Trata-se de uma livraria desenvolvida em C, que adapta para Python a interface de programação oficial (API) do sistema de bases de dados PostgreSQL, "libpq". Permite que programas "clientes" submetam pesquisas ("*queries*") a um servidor correndo uma base de dados PostgreSQL e recebam os resultados das mesmas. Suporta os principais tipos de dados da linguagem Python, e procede à sua compatibilização com os tipos do sistema de base de dados PostgreSQL (Di Gregorio, 2001).

Para criação de interface gráfica com o utilizador ("*GUI*"), utilizou-se a infraestrutura ("*framework*") Qt, versão 4.8, da empresa The Qt Company . Trata-se de conjunto de livrarias, desenvolvidas em C++, para permitir, entre outras funcionalidades, a utilização de

interfaces gráficas em diferentes sistemas operativos, nomeadamente Windows e Linux (Riverbank Computing Ltd., 2015). Para poderem ser utilizadas em conjunto com a linguagem Python, foi utilizada ainda a livraria PyQT4 da empresa Riverbank Computing Limited (que implementa "*bindings*" para Python) (Summerfield, 2007). A livraria Qt 4.8 possui licença do tipo LGPL, e a livraria PyQt4 do tipo GPL. A infraestrutura Qt foi concebida com os seguintes conceitos base:

- Abstração da interface gráfica (programação de janelas, menus e "*widgets*" independente do sistema operativo, com adaptação automática a cada);

- Implementação de mecanismo de eventos ("*signals*") e receptores ("*slots*") em que os objectos componentes da interface gráfica podem enviar mensagens com informação de eventos a outros componentes, que as recebem através de funções especiais ("*slots*") (Bodnar, 2011);

Finalmente, para visualização de dados espaciais em mapa, foi utilizada a API ("*Application Programming Interface*") do programa QGIS, na sua versão 2.10.1 (WIndows e Linux). Dada a inclusão de interface para programação da API em Python, é possível utilizar funcionalidades SIG, asseguradas pela aplicação QGIS, directamente a partir do programa de gestão de dados (QGIS Project , 2015, pág. 3 -43).

Tanto a aplicação QGis 2.10.1 (baseada em Qt) como a livraria PyQt4 utilizam a mesma ferramenta de geração de código (SIP) para assegurar a comunicação entre o interpretador Python e a estrutura Qt (C++).

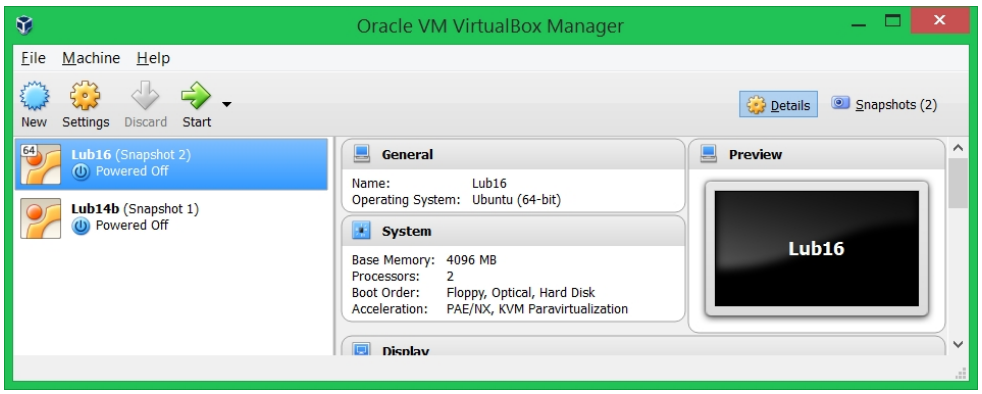
3.2.2. Estrutura do programa de gestão de dados

Procurou-se desde o início, conceber um programa com uma estrutura simples, com um conjunto de funcionalidades focado nos objectivos do projeto, utilizando soluções pragmáticas para resolução dos diferentes problemas e dificuldades encontrados. A filosofia seguida foi a de conseguir implementar um programa funcional, que permitisse testar o mecanismo de agregação de dados Pointcloud utilizando conjuntos de dados reais e, em simultâneo, criar registos de metadados de acordo com as normas SeaDataNet, da União Europeia. Poderão ser efectuadas optimizações ao programa em fase posterior, após período de utilização significativo.

O desenvolvimento do programa e da base de dados espaciais subjacente foi efectuado predominantemente em computador pessoal com sistema operativo Windows 8.1 Pro, por questões eminentemente práticas.

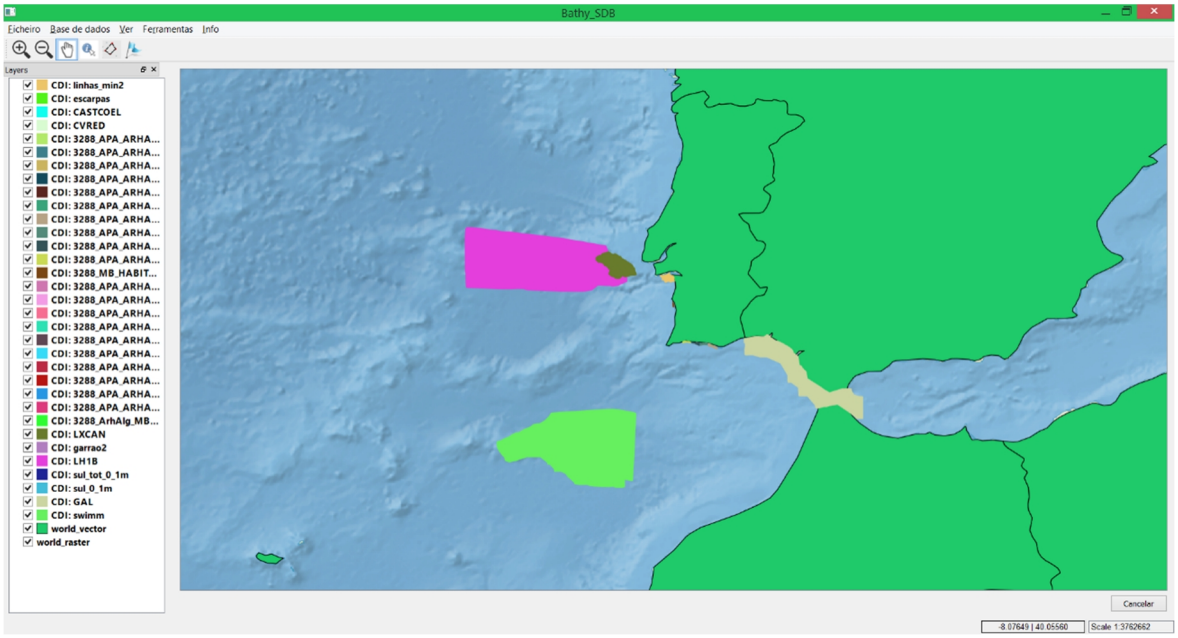
Recorrendo ao software VirtualBox, da empresa Oracle, foram criados dois computadores virtuais com sistema operativo Linux (Lubuntu 14 e Lubuntu 16), recriando uma rede (virtual) com um servidor de dados (PostGIS, com base de dados idêntica à da versão Windows, a correr em Lubuntu 14) e um cliente (o mesmo código Python da aplicação para Windows, com as livrarias correspondentes Qt 4.8 + PyQt4, a correr em Lubuntu 16). Sempre que se procederam a desenvolvimentos significativos em ambiente Windows estes foram replicados e testados na "rede" cliente-servidor Linux, sem dificuldades de maior (Figura 3.5).

Figura 3.5 – *Computadores virtuais Linux (Lubuntu 14 e 16)*



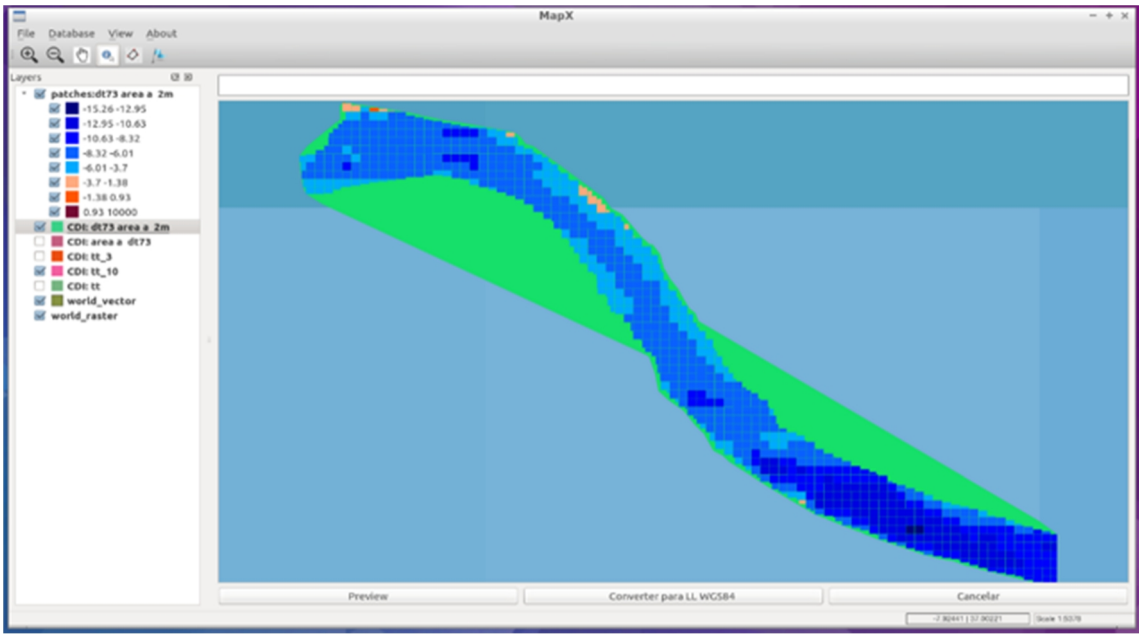
O programa é constituído por um módulo principal (*main.py*), o primeiro a ser executado e a partir do qual são acedidas todas as funcionalidades. Procede também à gestão dos eventos e diferentes menus da janela principal do programa, (correspondente à classe definida em "*main_window.py*"). É através da interação com os elementos (menus, botões) desta janela que se acedem às diferentes funcionalidades implementadas. A janela principal permite a visualização de mapa global, sobre o qual são apresentados poligonos representativos dos diferentes conjuntos de dados. É possível navegar no mapa com o auxílio do rato e proceder a operações de ampliação ou redução ("*zoom*") directamente com a roda do rato. Na figuras abaixos indicadas apresenta-se cópia de écran com a janela principal visível, apresentando mapa e tabela de conteúdos (Figuras 3.6, 3.7 e 3.8).

Figura 3.6– *Janela principal do programa de gestão de dados*



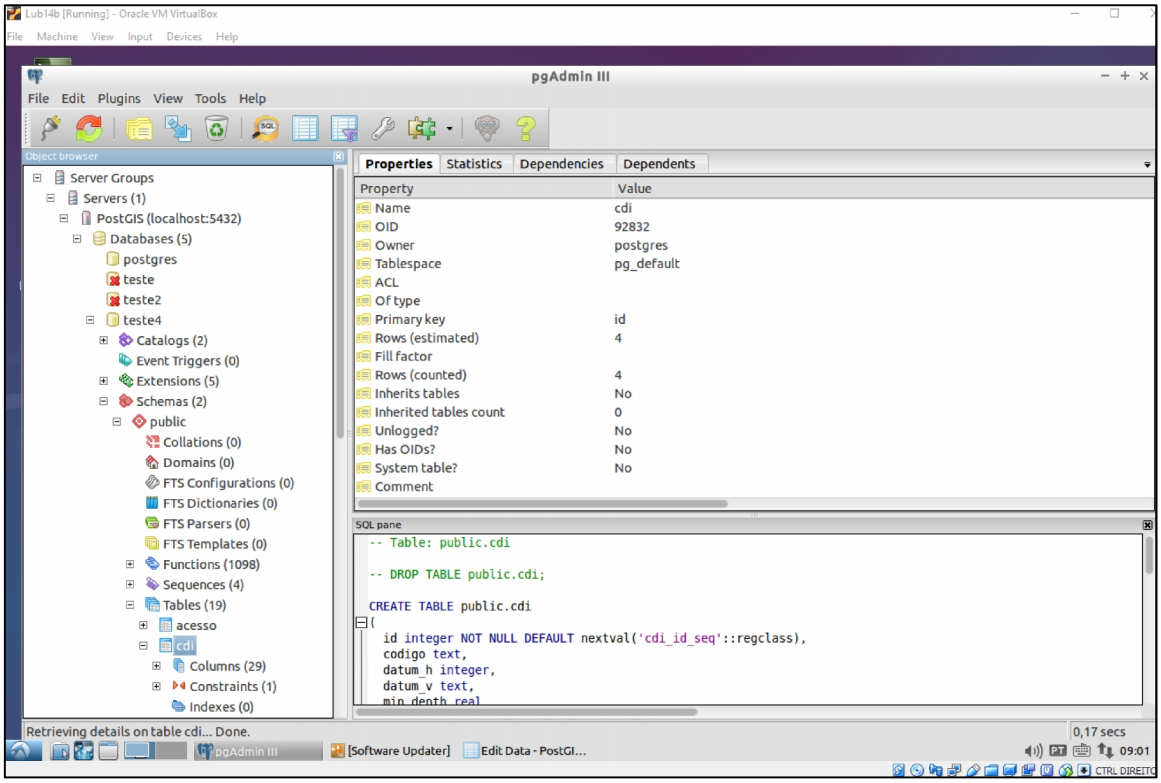
Nota. Visualização de mapa global e lista de "*datasets*" já introduzidos na base de dados.

Figura 3.7 –*Programa cliente em maquina virtual Lubuntu 16*



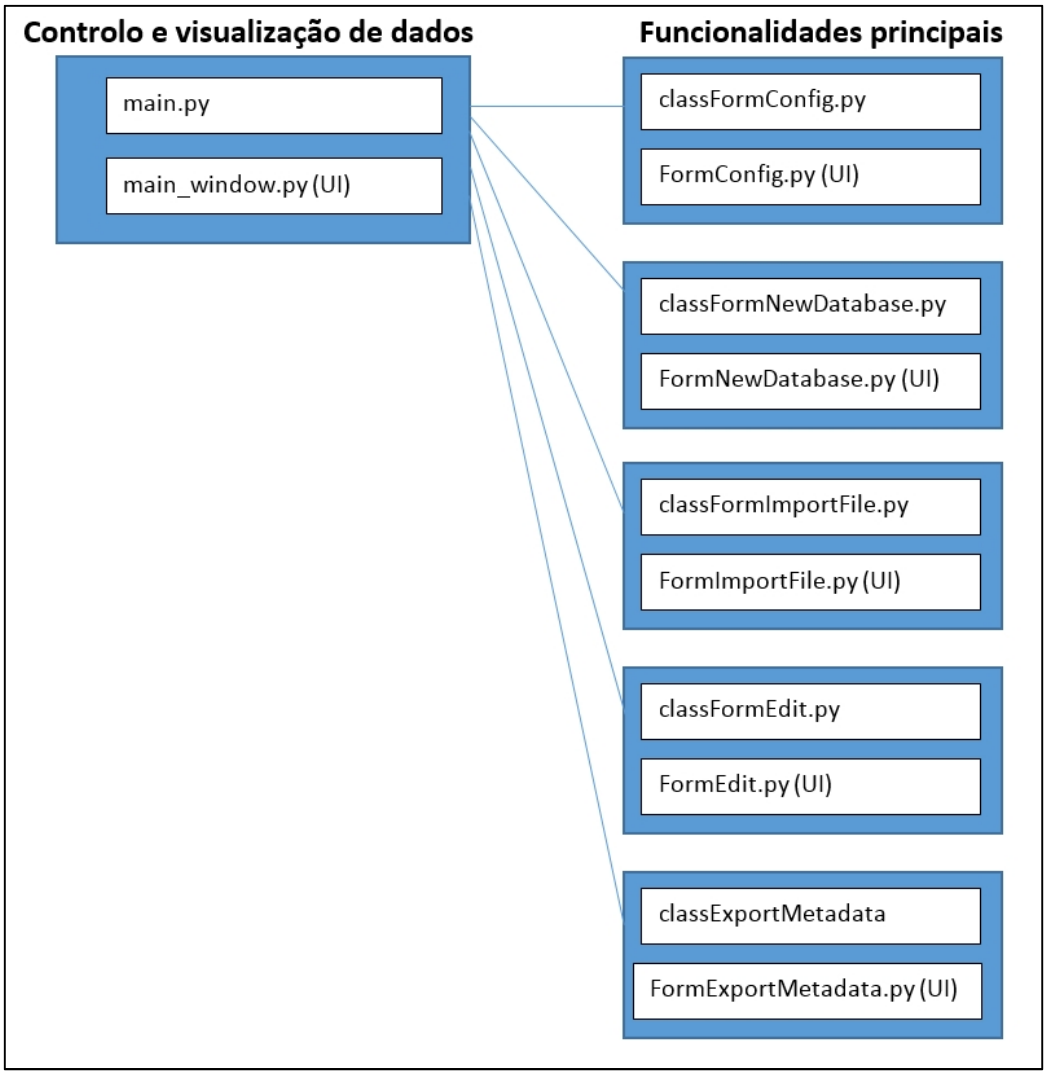
Nota. Mapa de parte do canal de Faro, com visualização de "*Pcpatches*".

Figura 3.8– *Servidor de dados em maquina virtual Lubuntu 14*



Todos os principais módulos da aplicação consistem numa "dupla" integrando uma classe que gere uma janela ("*form*") de interface com o utilizador, (para a qual é gerado código python pelas livrarias Qt/PyQt) . Na representação da figura 3.9, os ficheiros de código correspondentes a "*forms*" têm a anotação "(UI)" .

Figura 3.9 – *Esquema dos principais módulos do programa*



Nota. Classes e "*forms*" para interface com utilizador (UI).

Foi criado repositório na plataforma Github para todo o código desenvolvido para o programa. Este repositório está disponível para consulta na página Web https://github.com/jubasky/Bathy_SDB .

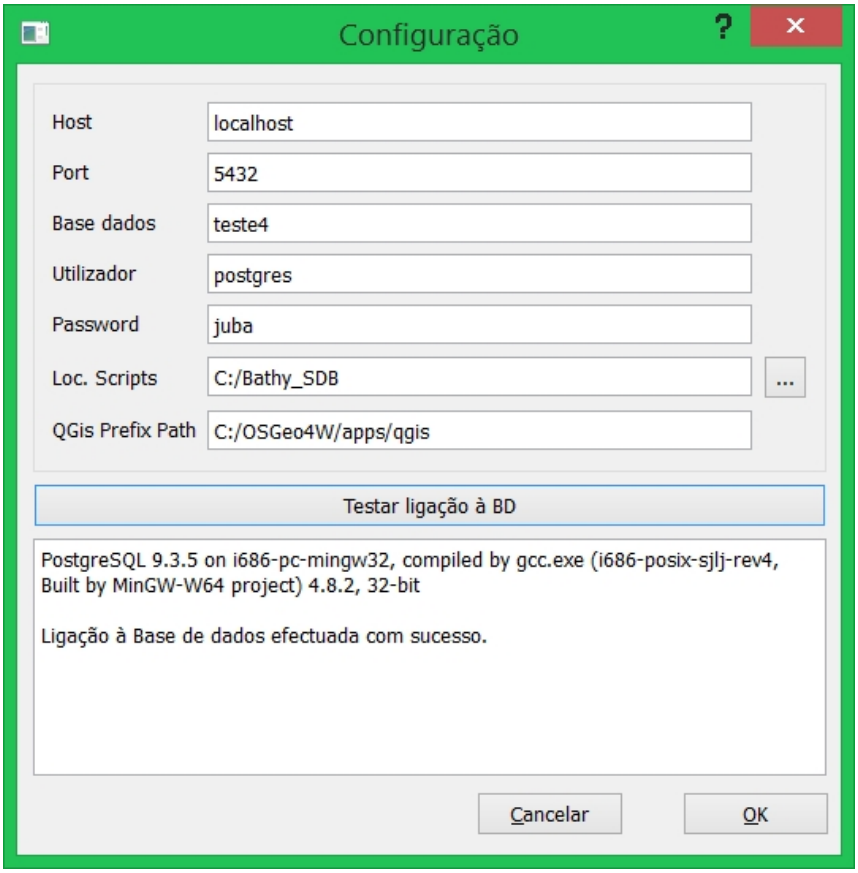
Nos pontos seguintes descrevem-se as principais funcionalidades do programa.

3.2.3. Funcionalidades principais

3.2.3.1. Configuração do programa

Para configuração dos parâmetros necessários para comunicação com o servidor de dados ("*Host*") e com a base de dados pretendida, e para identificação dos caminhos para pastas de "scripts" SQL e de dados gerados pelo programa, foi criado a janela "*FormConfig.py*", gerida pela classe "*classFormConfig.py*". Incluiu-se também a funcionalidade de testar a ligação parametrizada. Os dados preenchidos nesta "*form*" são gravados em ficheiro de texto com o nome "config.ini", localizado no mesmo directório do programa (Figura 3.10).

Figura 3.10– *Janela correspondente ao módulo "formConfig.py"*



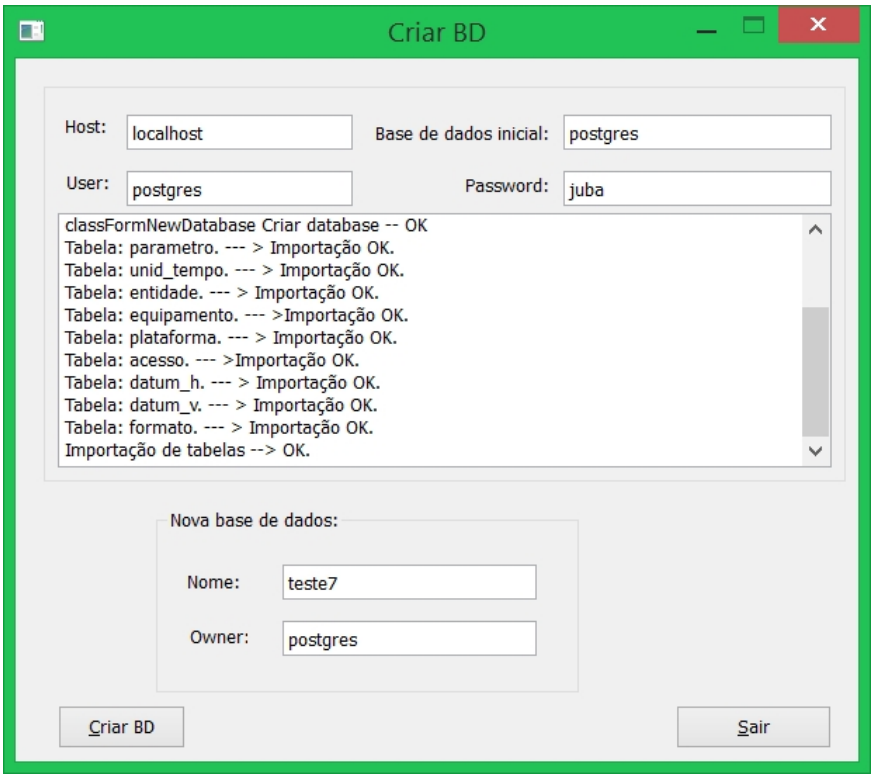
Nota. Módulo de configuração do programa.

3.2.3.2. Criação de base de dados

De forma a automatizar a criação de uma base de dados inicial, devidamente estruturada com todas as tabelas e "*views*" necessárias e pronta a receber dados, foram criados "*scripts*" SQL, gravados sob a forma de ficheiro de texto em pasta definida na configuração do programa ("C:\Bathy_SDB\Scripts", por defeito). Estes "*scripts*" são executados a partir de uma base de dados pré-existente, criada directamente com a aplicação PgAdmin 3, e que serve apenas para este fim, não necessitando de ser modificada pelo utilizador (Riggs, 2010, pág. 7-32). O módulo "*classFormNewDatabase.py*", em conjunto com a janela definida em "*FormNewDatabase.py*" procede à execução dos diferentes "*scripts*", de forma sequencial.

O "*script*" "*sql_create_db.sql*" procede à criação de nova base de dados com o nome definido pelo utilizador, executa as instruções para implementação das extensões PostGIS e Pointcloud, define a estrutura "*pointcloud*" a utilizar, seguindo-se a definição das diferentes tabelas e respectivos campos. As tabelas auxiliares são preenchidas pelo programa, aparir de ficheiros de texto com códigos e descrições (vocabulários) disponibilizados pelo estrutura SeaDataNet (Figura 3.11). O "*script*" "*sql_create_db.sql*" descrito acima é apresentados em anexo (Anexo A –"*Scripts SQL*").

Figura 3.11 – *"Form" definida pelo módulo "formNewDatabase.py*



Nota. Módulo para criação de nova base de dados.

3.2.3.3. Importação de dados

Para introdução de dados e metadados foi criada janela específica, "*formImportFile.py*" e respectiva classe de controlo "*classFormImportFile.py*". Os dados de batimetria/altimetria com ou sem informação de "*backscatter*" acústico e de cor (RGB) são importados a partir de ficheiros de texto, com os valores para cada ponto agrupados por linhas e separados por vírgula, ponto e vírgula ou por espaço.

Os dados podem ser submetidos em diferentes sistemas de coordenadas e com diferentes data verticais. As coordenadas horizontais são convertidas para longitudes e latitudes WGS84 (EPSG: 4326)) antes de serem importadas, recorrendo às livrarias Proj4 e pyproj ("*bindings*" para Python).

O datum vertical é mantido tal como o operador o definiu. Preferencialmente, os dados de profundidade para zonas costeiras deverão estar referidos a um datum compativel com LAT ("*Lowest Astronomical Tide*"), por exemplo ao Zero Hidrográfico, no caso de Portugal continental. Os dados de profundidade para levantamentos de alto mar deverão ser referenciados ao Nível Médio do Mar ("*Mean Sea Level*").

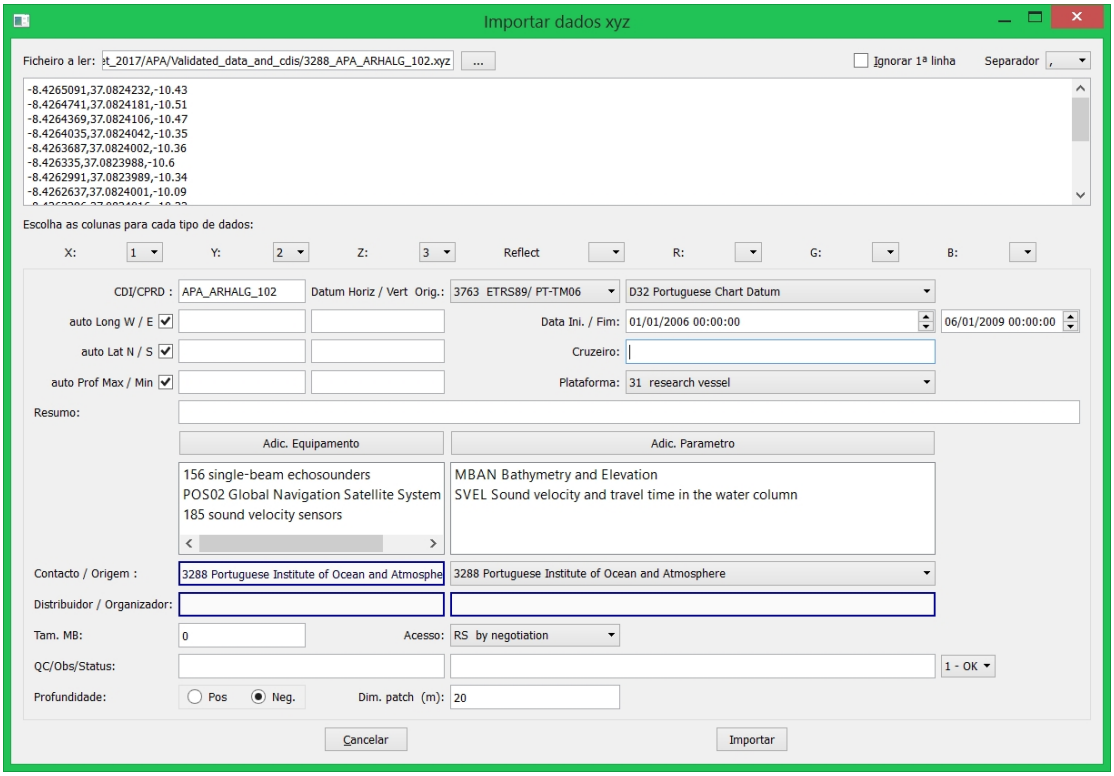
A "*form*" de introdução de dados considera todos os campos obrigatórios definidos pelo projeto Emodnet para metadados associados a "*datasets*" de batimetria.

São disponibilizados através de "*widgets*" do tipo "*ComboBox*" as listas de atributos definidos segundo os vocabulários SeaDataNet para os campos adequados (equipamentos utilizados, parâmetros medidos, entidades), lidas a partir das tabelas auxiliares .

Note-se que o campo CDI (utilizado aqui para designar o código alfanumérico do cruzeiro/campanha) tem que ser único na base de dados local, dado ser utilizado como chave primária na base de dados pan-europeia do projeto Emodnet (que integra toda a informação de metadados de batimetria disponibilizada pelos diferentes parceiros). O módulo "*classFormImportFile.py*" verifica este quesito antes de iniciar a importação.

A dimensão máxima das estruturas "*Pcpatch*" (em metros) a serem criadas no processo de importação de dados é definida pelo utilizador utilizando o campo "Dim. Patch" (Figura 3.12).

Figura 3.12 – *Janela do módulo de importação de dados "FormImportFile.py"*



Após o preenchimento de todos os campos, o utilizador pode iniciar o processo de importação (botão "importar). Este processo é efectuado em 7 passos:

- Inserção de registo de metadados na tabela "CDI", registando o valor da chave primária deste registo (gerada automáticamente pelo SBDE, e gravada no campo "*id*"). São também inseridos os registos necessários nas tabelas de ligação "Cdi_equip" e "Cdi_param";

- Gravação de ficheiro temporário com os dados importados devidamente formatados (longitude e latitude, em graus decimais (WGS84) com 7 casas decimais, profundidade em metros, com três casas decimais, valor de retrodispersão ("*backscatter*") como inteiro, e valores de R,G,B, como inteiros de 0 a 255);

- Inserção, em bloco, dos valores do ficheiro anterior em tabela temporária na base de dados ("pontos_temp"), com campos ("x", "y", "z", "i", "r," "g", ,"b") com os tipos adequados ("*double precision*", "*single precision*", "*integer*" e "*smallint*");

- Conversão das linhas da tabela anterior para o formato "*pointcloud*" (definido na tabela "*pointcloud_formats*" recorrendo à função "*PC_MakePoint*", disponibilizada pela

extensão "*Pointcloud*", e inserção das linhas resultantes na tabela temporária "*pc_points*" . O parágrafo abaixo mostra a "*query*" SQL utilizada para este fim.

"INSERT INTO pc_points (pt) SELECT PC_MakePoint(3, ARRAY[a,b,c,intensity,r,g,b]) FROM (SELECT x as a, y as b, z as c, i AS intensity, rr as r, gg as g, bb as b FROM pontos_temp) AS values;"

- Agregação dos registos da tabela temporária "*pc_points*" em estruturas "*Pcpatch*", recorrendo à função "*PC_Patch*" (assegurada pela extensão "*Pointcloud*") e inserção dos resultados na tabela "patches", com recurso à "*query*" abaixo ilustrada.

"INSERT INTO patches (pa, cdi) SELECT PC_Patch (pt), -99 FROM pc_points group by st_snaptogrid(pt::geometry, -88, -88);"

Os valores -99 e -88 são substituídos programaticamente pelo "*id*" do "*dataset*" em causa e pelo tamanho do "*patch*", respectivamente.

- Inserção de registo na tabela "patches_info", com informação de síntese relativa ao "*dataset*" importado, constituída pelo "*id*", código descritivo, número total de pontos, profundidade média, mínima e máxima e geometria (polígono) da envolvente do conjunto de "*patches*" resultante.

"INSERT INTO patches_info (id, codigo, n, prof, prof_max, prof_min, geom) SELECT patches.cdi, -88, sum(PC_NumPoints(pa)) as n, avg(PC_PatchAvg(pa, 'z')) as prof, min(PC_PatchMin(pa, 'z')) as prof_max, max(PC_PatchMax(pa, 'z')) as prof_min, st_concavehull(st_collect(geometry(patches.pa)), 0.97::double precision) AS geom FROM patches WHERE patches.cdi = -99 group by patches.cdi;"

Os valores -88 e -99 são substituídos programaticamente pelo codigo descritivo do "*dataset*" importado (campo "codigo)" e pela sua chave primária (campo "id").

- Finalmente, procede-se à remoção dos registos das tabelas temporárias utilizadas ("pontos_temp" e "pc_points"), de forma a ficarem prontas para nova importação de dados.

3.2.3.4. Visualização da informação

A visualização da informação existente na base de dados é feita directamente na janela principal do programa ("*main_Window.py*"). Para esse fim, foi utilizada uma das principais componentes da API QGis, a classe "*QGisMapCanvas*", que permite a representação em mapa dos diferentes conjuntos, e ainda a componente "*QgsLayerTreeViewMenuProvider*" para a construção de tabela de conteúdos, na forma de lista do tipo "*TreeView*".

Quando o programa se inicia, é lida a tabela "patches_info" (com informação de síntese para cada conjunto de dados, incluíndo polígono envolvente) para povoamento do mapa e da tabela de conteúdos.

É possível proceder a operações genéricas de "*pan*" e "*zoom*" no mapa recorrendo ao rato do computador, bem como seleccionar um determinado "*dataset*" na tabela de conteúdos e proceder a operação de "*zoom*" selectivo. Foram implementadas ainda funcionalidades para visualização dos "*patches*" constituintes de um "*dataset*", bem como dos pontos individuais subjacentes.

Para visualização dos metadados pode ser utilizada a opção de edição de dados, que abre uma janela com este fim, através do menu de contexto da tabela de conteúdos.

As funcionalidades descritas acima são ilustradas nas figuras seguintes (Figuras 3.13 a 3.16).

Figura 3.13 - *Componentes para visualização de dados*

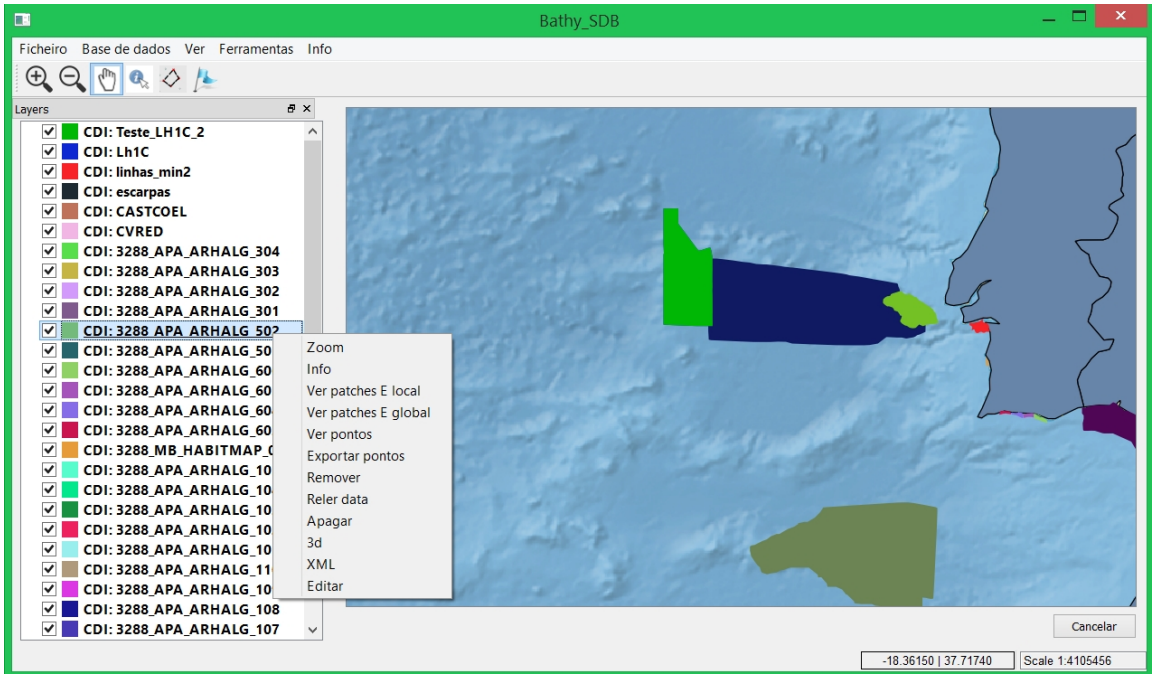


Figura 3.14- *Visualização de "patches" de um "dataset" selecionado*

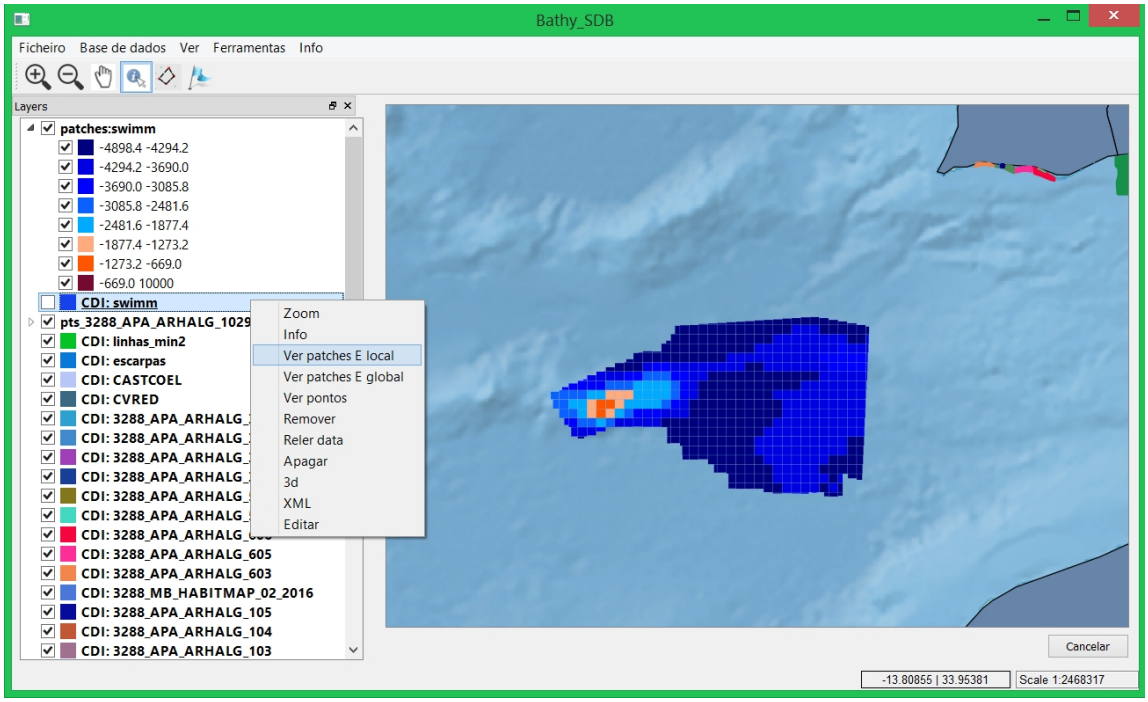


Figura 3.15 - *Visualização de pontos individuais*

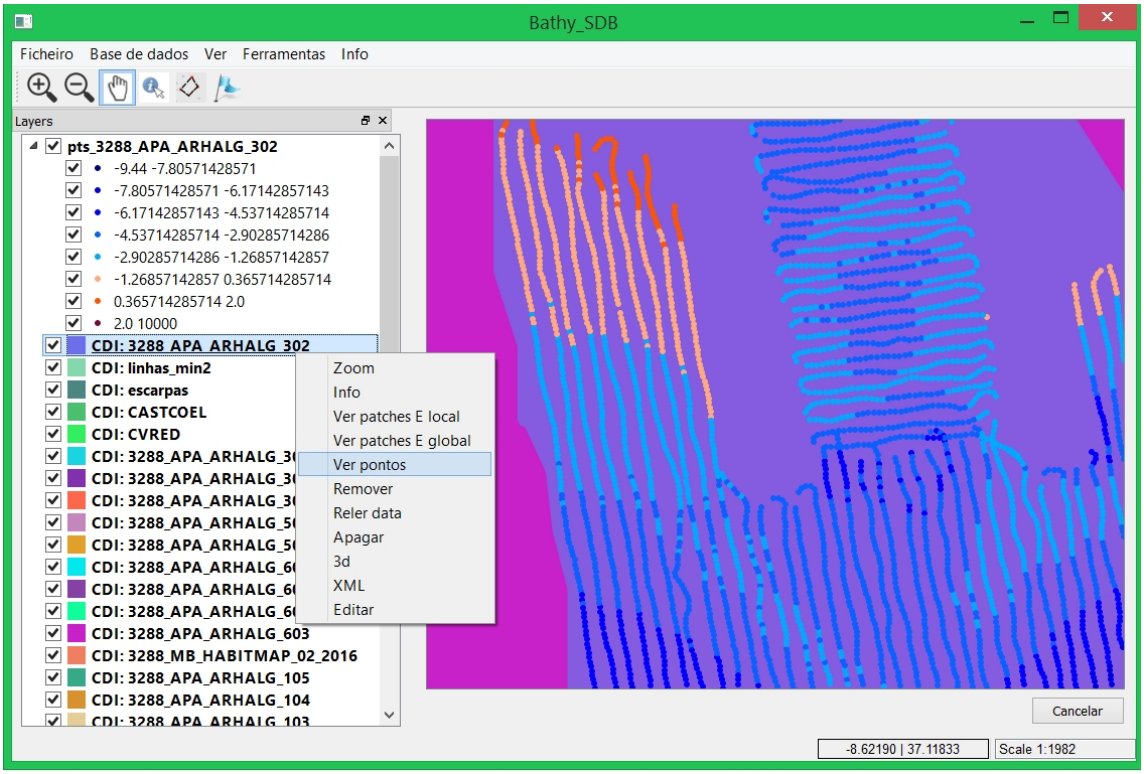
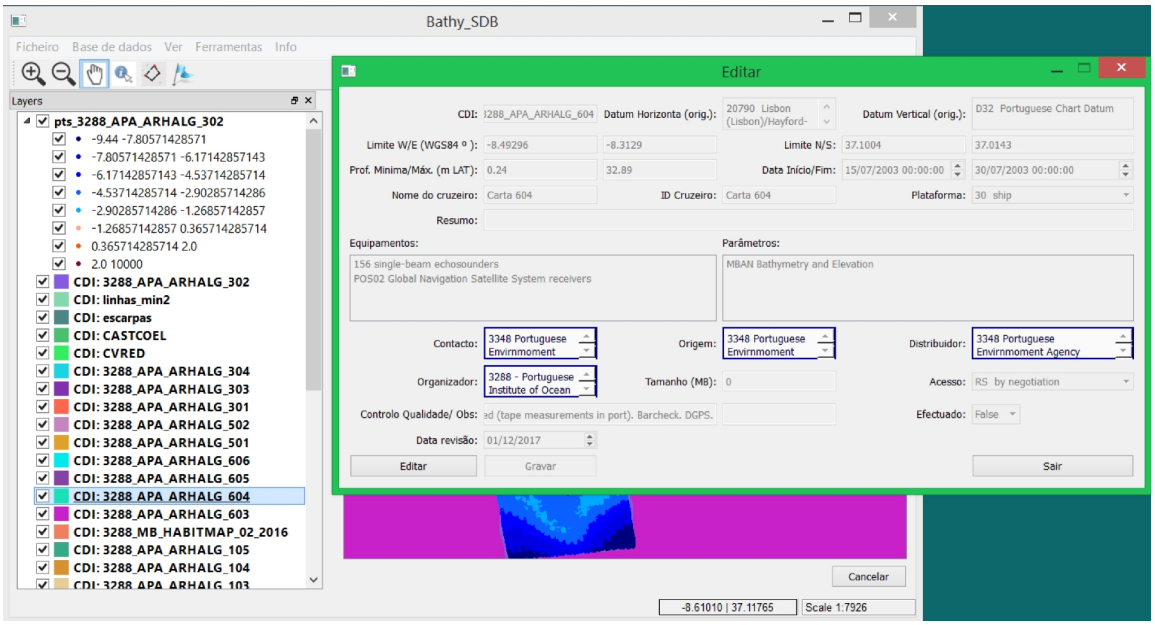


Figura 3.16 - *Visualização e edição de metadados de um "dataset"*



3.2.3.5. Pesquisas espaciais

De forma a possibilitar a pesquisa de dados por critérios espaciais foi construída ferramenta para que o utilizador possa desenhar diretamente no mapa da janela principal um rectângulo ("*rubber band*") para definição de "*bounding box*". Após esta ação é possível editar as coordenadas para um ajuste mais detalhado dos limites. Após validação dos limites, é efectuada uma "*query*" (pesquisa) espacial recorrendo à função "*PC_Intersects*" (da extensão Pointcloud) que recebe como argumentos as geometrias da tabela "*patches*" e o rectângulo definido pelo utilizador. O resultado da pesquisa ("*patches*" compatíveis com o critério de interseção) é gravado na tabela temporária "patches_sel", adicionado à tabela de conteúdos e visualizado em mapa.

Prevê-se complementar esta funcionalidade com a possibilidade de se adicionarem mais critérios relacionados com os metadados (datas, equipamentos utilizados, navios, parâmetros medidos e outros) além da "*bounding box*". As figuras 3.17 e 3.18, abaixo representadas, ilustram a funcionalidade descrita.

Figura 3.17 - *Pesquisa espacial – definição de rectângulo ("bounding box")*

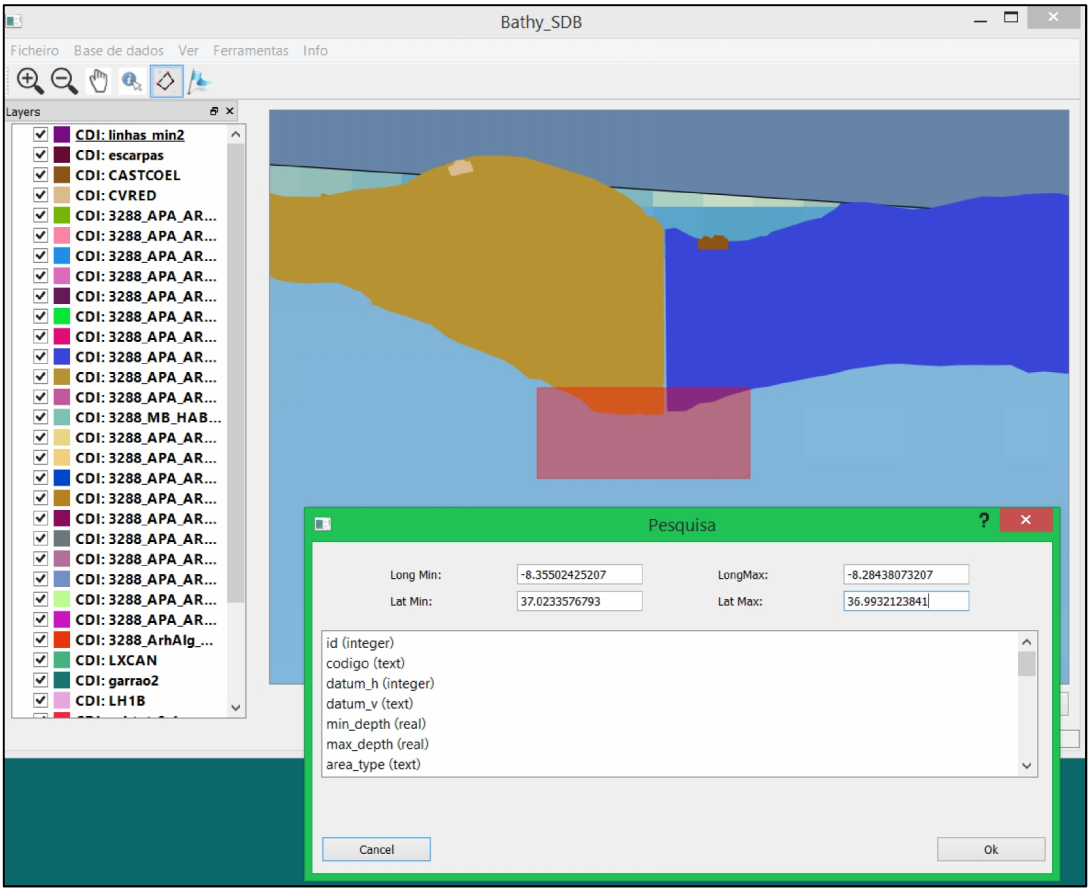
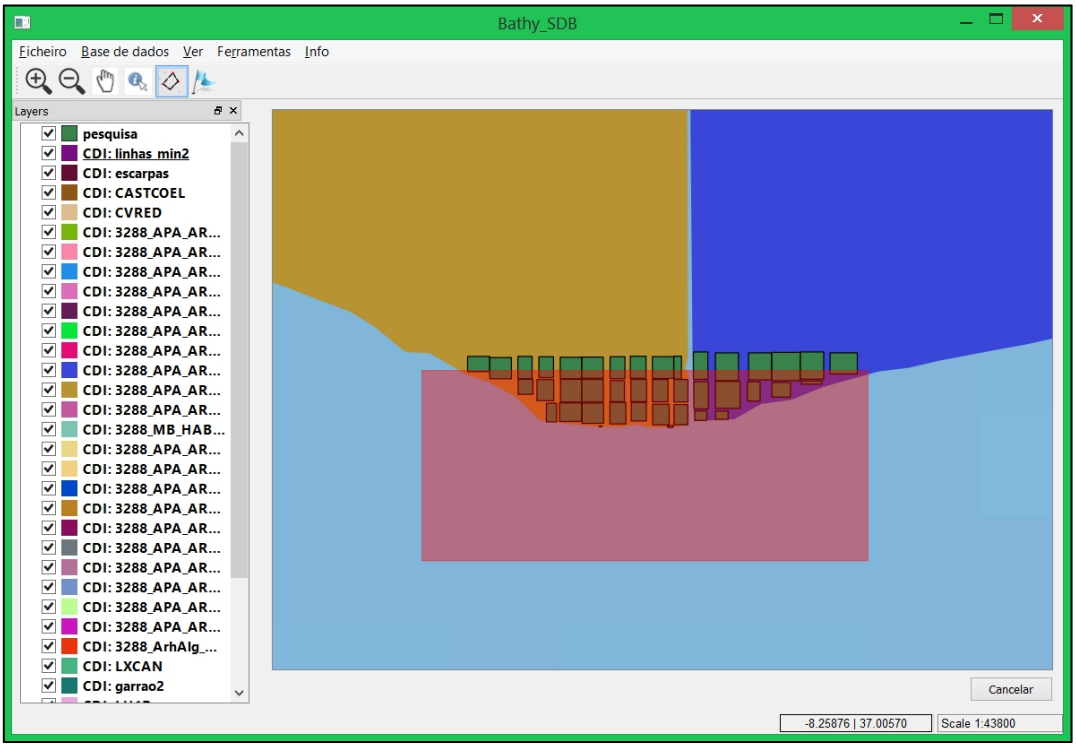


Figura 3.18- *Pesquisa espacial- visualização de resultados*



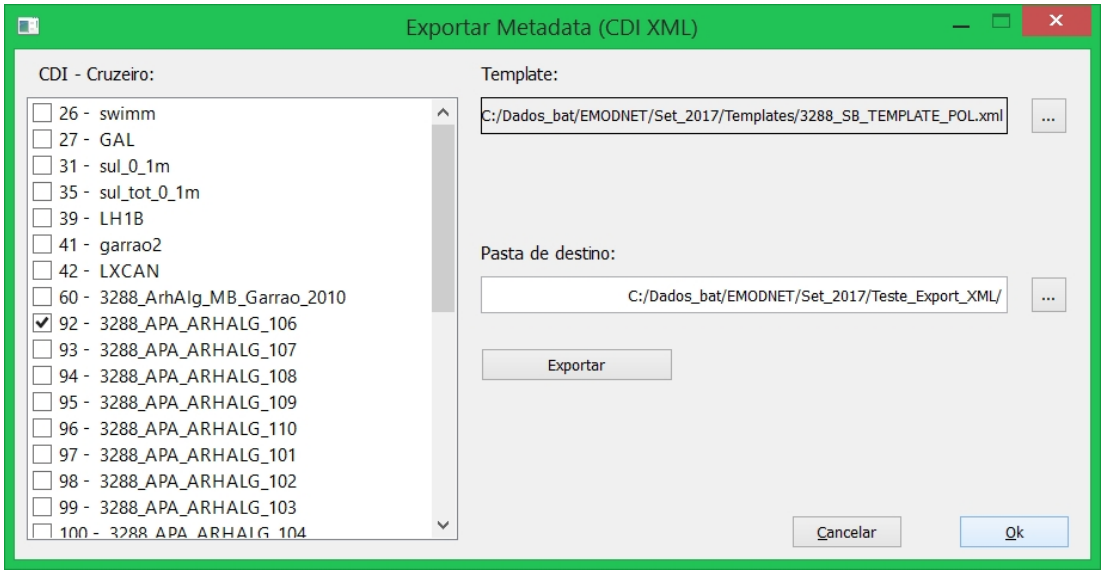
Nota. Visualização dos "*patches*" compatíveis com o critério de pesquisa.

3.2.3.6. Exportação de metadados

Para permitir a exportação dos metadados de cada "*dataset*" foi criada a janela "*formExportMetada.py*" e a respectiva classe de gestão "*classExportMetadata.py*". O utilizador pode escolher qual ou quais os "*datasets*" para os quais pretende exportar a informação.

Dado que esta função foi desenvolvida tendo em vista as necessidades específicas do projeto Emodnet Bathymetry, que requer um formato padronizado com um esquema XML associado, optou-se por criar um "*template*" (correspondente a um "*dataset*" fictício) gravado em disco. O utilizador seleciona o "template" a utilizar para o processo de exportação e a classe "*classExportMetadata.py*" procede à substituição dos valores para cada campo, gravando o resultado em novo ficheiro XML, com nome determinado pelo utilizador. Este mecanismo foi testado recorrendo à aplicação fornecida pelo projeto Emodnet para criação deste tipo de ficheiros de metadados, que incluí procedimentos de validação da informação introduzida (Figura 3.19).

Figura 3.19 - *Janela para exportação de metadados*



Nota. Dados exportados no formato CDI-XML, definido pelo projeto Emodnet Bathymetry.

.

4. RESULTADOS E DISCUSSÃO

4.1. Eficácia da metodologia Pointcloud

Foram efetuados diversos testes com importação de dados provenientes de levantamentos batimétricos efectuados com sistemas de feixe simples e multifeixe, não se encontrando dificuldades em concretizar o processo de importação tal como descrito nos pontos anteriores.

A importação de dados para o formato "*pointcloud*" e a sua agregação em "*Pcpatches*" é no entanto um processo moderadamente lento. A título de exemplo, um ficheiro de texto do tipo xyz com 30 MB (com aproximadamente 729 mil pontos) necessita de cerca de 20 segundos para ser convertido para o formato "*Pcpatch*" e inserido na respectiva tabela da base de dados. Este tempo incluí a duração do processo de criação do respectivo registo na tabela "patches_info" (com informação de síntese e com a geometria da envolvente de todos os "*patches*"). Note-se que para a construção desta envolvente é utilizada a função PostGIS "*st_concavehull*", para melhor definição da área correspondente ao levantamento/cruzeiro.

A utilização de compressão inerente à metodologia "*pointcloud*" (compressão dimensional) permite obter "*ratios*" entre 1:3 a 1:5 quando os "*patches*" correspondem a áreas onde as diferentes dimensões apresentam reduzida variabilidade (Ramsey, 2012).

O facto de ser possível visualizar em mapa, de forma simples e rápida, o conjunto de "*patches*" correspondente a um levantamento, com escala de cores, é uma vantagem desta metodologia. Desta forma, o utilizador pode ter uma primeira avaliação da morfologia da área correspondente evitando a demora associada à representação de centenas de milhares de pontos individuais no mapa da janela principal.

Para além destes factores, a agregação da informação em estruturas "*Pcpatch*" possibilita que cada linha da tabela "*patches*" (principal repositório da informação altimétrica/batimétrica) armazene até 600 pontos individuais, uma solução muito mais racional e eficiente que o armazenamento de um ponto individual em cada linha de uma tabela "clássica". Esta agregação possibilita também a execução de pesquisas espaciais mais rápidas.

4.2. Cumprimento dos objectivos

Com a implementação de um SBDE em PostgreSQL/PostGIS, recorrendo à livraria "*pointcloud*" para armazenamento dos dados de base e com a utilização de software livre para desenvolvimento do programa de gestão, considera-se que se alcançaram os 5 objectivos principais definidos para o presente projeto:

- Concepção e implementação de estruturas de dados que permitam o armazenamento eficaz dos vários tipos de informação batimétrica e altimétrica disponíveis. A utilização de estruturas "*Pcpatch*" permitiu a concretização deste objectivo, como explicado no ponto anterior;

– Concepção e implementação de conjunto de tabelas auxiliares para integração de metadados. O programa desenvolvido concretizou este objectivo. Foram desenvolvidas ferramentas para criação das tabelas necessárias e para o seu preenchimento automático a partir dos vocabulários definidos pelo organismo SeaDataNet da União Europeia (ponto 4.1.3 do presente trabalho);

– Desenvolvimento de aplicação de gestão e manutenção da base de dados que permita a importação de dados e a sua devida integração nas estruturas escolhidas ("*Pcpatch*"). Conforme foi mostrado no ponto 4.2.3.3 desta tese, foram desenvolvidas as funcionalidades necessárias à importação de dados de batimetria e altimetria bem como à criação dos metadados necessários ao cumprimento das normas de partilha de dados do projeto Emodnet Bathymetry, bem como funções específicas para edição de metadados e eliminação de conjuntos de dados (e respectivos metadados), caso seja necessário;

- Desenvolvimento de aplicação com interface gráfica para visualização dos dados e construção de pesquisas ("*queries*") espaciais ou por atributos. Conforme foi explicado nos pontos 4.2.3.4 e 4.2.3.5 (visualização e pesquisas espaciais) este objectivo foi também concretizado no essencial, faltando apenas completar a implementação de pesquisa por atributos, funcionalidade que se considera relativamente trivial;

– Desenvolvimento de ferramentas de exportação de dados e metadados para os formatos necessários à integração nas infra-estruturas europeias de divulgação e partilha de dados em que o IPMA participa. De acordo com o exposto no ponto 4.2.3.6 (exportação de metadados) este objectivo foi cumprido, recorrendo à utilização de "*templates*" para construção dos ficheiros XML no formato especificado pelo projeto Emodnet Bathymetry. A exportação dos dados batimétricos é feita diretamente a partir do menu de contexto da tabela de conteúdos da janela principal do programa, para ficheiro de texto do tipo "xyz";

4.3. Dificuldades encontradas

As principais dificuldades encontradas durante o desenvolvimento do projeto, tiveram a ver com o mecanismo de agregação de pontos ("*pointcloud points*") em "*Pcpatches*". Dado que são possíveis várias formas de agregação, foi testada em primeiro lugar uma forma que considera apenas um número fixo de pontos por "*patch*", que seria, em princípio, a solução ideal. No entanto, esta forma apenas funciona devidamente se os pontos estiveram já ordenados espacialmente, isto é, com uma distribuição sequencial contígua , o que não acontece na maior parte dos casos, ocasionando a geração de "*Pcpatches*" com dimensões muito variáveis, com formas rectangulares por vezes extremamente alongadas, que não possibilitam uma visualização explícita da informação armazenada. Optou-se por uma forma de agregação em que se define directamente a dimensão do rectângulo correspondente aos diferentes "*Pcpatches*" a criar, que apresenta a vantagem de gerar formas muito regulares, de igual dimensão.

Esta opção provoca que no processo de importação de dados se tenha que proceder a alguns ensaios, com diferentes tamanhos, de forma a limitar o número de pontos ao máximo de 600, número a partir do qual se ultrapassa o máximo possível para armazenar um único registo de dados numa tabela sem se recorrer a mecanismo de utilização de apontadores para tabelas auxiliares criadas pelo SBDE de forma automática.

4.4. Limitações do projeto

O projeto aqui apresentado restringe-se a um conjunto relativamente limitado de funcionalidades, com o objectivo de se conseguir concretizar uma solução de armazenamentos de dados de altimetria/batimetria que seja ao mesmo tempo simples de explorar e de manter.

Considera-se que a solução apresentada no presente trabalho é adequada para armazenamento de informação de batimetria a adquirir pelo IPMA nos próximos anos. A versão do SBDE utilizada (PostgreSQL/PostGIS 9.3.5) apresenta o limite de 32 Terabytes para o tamanho de tabelas, o que deverá ser mais que suficiente para os próximo 10 anos. A versão mais recente do sistema de bases de dados PostgreSQL (v11) apresenta já a possibilidade de se trabalhar com partições de forma eficiente, que permitem a construção de uma tabela virtual, composta por múltiplas tabelas de estrutura idêntica. Assim, o limite de 32 TB é eliminado, passando o tamanho máximo para um valor da ordem de 2 Exabytes (2 milhões de TB).

4.5. Evolução futura

Qualquer programa activamente utilizado necessita de manutenção, quer seja para correcção de erros, como para desenvolvimento de novas funcionalidades. Prevê-se que seja necessário efetuar correcções ao programa actual (resolução de "*bugs*") bem como desenvolver novas funções, nomeadamente um processo de determinar automáticamente a dimensão adequada das estruturas "*Pcpatch*" para armazenamento de dados provenientes de levantamentos com diferentes resoluções espaciais. Será também necessário prever actualizações tanto da base de dados PostgreSQL (para novas versões, com mais funcionalidades) como da linguagem de programação e livrarias utilizadas. A evolução mais premente consiste na mudança da liguagem de programação para Python 3, dada que a versão utilizada neste projeto (2.7) deixará de ser mantida em 2020. Esta evolução deverá ser acompanhada pela adoção das livrarias Qt 5 e PyQt5 de forma a manter o projeto actual.

5. CONCLUSÕES

Considera-se que o projeto aqui apresentado constitui um bom corolário da aprendizagem efectuada durante o Mestrado em Geomática. Foi possível colocar em prática o conhecimento adquirido, de uma forma real, na resolução de uma necessidade concreta. A execução do projeto agora apresentado levou ainda à necessidade de adquirir mais conhecimentos e lidar com novas formas de armazenamento e exploração de dados espaciais, para além daquelas com que o autor estava familiarizado. O produto final, embora concerteza não isento de erros e imperfeições, cumpre com os objetivos definidos à partida e constitui, desde já uma ferramenta útil para o fim a que se destina, a manutenção de dados e metadados de batimetria de uma forma integrada, com ferramentas adequadas aos requisitos principais decorrentes da participação da minha instituição (IPMA, IP) no projeto de partilha de dados Emodnet Bathymetry.

Para manter este projeto funcional, será necessário, a curto prazo, adoptar novas versões de SBDE e da liguagem de programação e livrarias utilizadas, para o que, dada a experiência entretanto adquirida, não se prevêm dificuldades inultrapassáveis.

6. BIBLIOGRAFIA

Bodnar, J. (2011). *Events and Signals in PyQt4*. Acedido em: 17, Janeiro, 2017, em: http://zetcode.com/gui/pyqt4/eventsandsignals/

Corti, P., Kraft, T.J., Mather, S.V & Park, B. (2014). *PostGIS Cookbook*. Packt Publishing Ltd. Disponível em: https://www.packtpub.com/big-data-and-business-intelligence/postgis-cookbook.

Di Gregorio, F. & Varrazzo, D. (2001). *Psycopg – PostgreSQL database adapter for Python*. Acedido em 19,1,2017, em: http://initd.org/psycopg/docs/

Güting, R.H. (1994). An introduction to spatial database systems. *The International Journal on Very Large Data Bases*, 3 (4), pág. 357-399. Disponível em: https://dl.acm.org/doi/10.5555/615204.615206

Heyer, T., Hiesinger, H., Reiss, D., Erkeling, G., Bernhardt, H., Luesebrink, D., Jaumann, R. (2018). The multi-temporal database of planetary image data (MUTED): A web-based tool for studying dynamic Mars. *Planetary and Space Science*, 159, pág. 56-65. Disponível em: https://www.sciencedirect.com/science/article/pii/S0032063317304038.

QGIS Project (2015) - *PyQGIS developer cookbook*. Acedido em 19,1,2017, em: https://qgis-docs.readthedocs.io/en/latest/docs/pyqgis_developer_cookbook/index.html

Riggs, S., Krosing, H. (2010). *PostgreSQL 9 Administration Cookbook*. Packt Publishing Ltd. Disponível em: https://itbook.download/topic/10150.

Ramsey, P. (2012). *A PostgreSQL extension for storing point cloud (LIDAR) data*. Acedido em: 3, Julho, 2016, em: https://github.com/pgpointcloud/pointcloud.

Riverbank Computing Ltd & the Qt Company (2015) - *PyQt Class Reference*. Acedido em 25/10/2016, em: https://www.riverbankcomputing.com/static/Docs/ PyQt4/ classes.html

Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Disponível em: http://www.qtrac.eu/pyqtbook.html.

The PostgreSQL Global Development Group (2014). *PostgreSQL 9.3 Documentation*. Acedido em: 4, Julho, 2016, em: https://www.postgresql.org/docs/9.3/index.html.

ANEXOS

7. ANEXO A. "*SCRIPTS*" SQL

"*Script*" CreateDatabaseSql.sql

Este "*script*" procede à criação das extensões necessárias para funcionalidades espaciais (PostGIS) e estruturas "*poincloud*". Define a estrutura "*PcPoint*" e cria as tabelas principais e auxiliares, bem como as relações entre elas. São ainda definidas as "*views*" e funções necessárias.

1 - Criação das extensões necessárias para funcionalidades espaciais (PostGIS) e estruturas "*poincloud*"

```
/* ------------------------------------------------------------ Criar extensões à base de dados  */
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION pointcloud;
CREATE EXTENSION pointcloud_postgis;
/* ------------------------------------------------------------------------ */
```

2 - Definição da estrutura "PcPoint" a utilizar

```
/* ------------------------------------------------------------ Criar formato para PcPoint  */
INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (3, 4326, '<?xml
version="1.0"  encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <pc:dimension>
  <pc:position>1</pc:position>
  <pc:size>4</pc:size>
  <pc:description>X : Longitude wgs84 como inteiro de 4 bytes, com escala de 10^(-
   7)</pc:description>
  <pc:name>X</pc:name>
  <pc:interpretation>int32_t</pc:interpretation>
  <pc:scale>0.0000001</pc:scale>
 </pc:dimension>
 <pc:dimension>
  <pc:position>2</pc:position>
  <pc:size>4</pc:size>
  <pc:description>Y : Latitude wgs84 como inteiro de 4 bytes, com escala de 10^(-7).
```

```
    </pc:description>
    <pc:name>Y</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.0000001</pc:scale>
</pc:dimension>
  <pc:dimension>
    <pc:position>3</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Z: profundidade como inteiro de 4 bytes, com escala de 10^(-
     4).</pc:description>
    <pc:name>Z</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.0001</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>4</pc:position>
    <pc:size>2</pc:size>
    <pc:description>Reflectividade</pc:description>
    <pc:name>Ref</pc:name>
    <pc:interpretation>uint16_t</pc:interpretation>
    <pc:scale>1</pc:scale>
 </pc:dimension>
  <pc:dimension>
    <pc:position>5</pc:position>
    <pc:size>1</pc:size>
    <pc:description>Red.</pc:description>
    <pc:name>R</pc:name>
    <pc:interpretation>uint8_t</pc:interpretation>
    <pc:scale>1</pc:scale>
  </pc:dimension>
<pc:dimension>
    <pc:position>6</pc:position>
    <pc:size>1</pc:size>
    <pc:description>Green.</pc:description>
    <pc:name>G</pc:name>
    <pc:interpretation>uint8_t</pc:interpretation>
    <pc:scale>1</pc:scale>
  </pc:dimension>
<pc:dimension>
    <pc:position>7</pc:position>
    <pc:size>1</pc:size>
    <pc:description>Blue.</pc:description>
    <pc:name>B</pc:name>
    <pc:interpretation>uint8_t</pc:interpretation>
    <pc:scale>1</pc:scale>
  </pc:dimension>
  <pc:metadata>
    <Metadata name="compression">dimensional</Metadata>
  </pc:metadata>
</pc:PointCloudSchema>');
```

3 – Criação das tabelas auxiliares

3.1 Tabela Acesso

```
-- Table: public.acesso
-- DROP TABLE public.acesso;
CREATE TABLE public.acesso
(
  id text NOT NULL, -- Codigo alfa numerico EDMO
  acesso text, -- Descricao do acesso
  CONSTRAINT acesso_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.acesso
  OWNER TO postgres;
COMMENT ON COLUMN public.acesso.id IS 'Codigo alfa numerico EDMO';
COMMENT ON COLUMN public.acesso.acesso IS 'Descricao do acesso';
```

3.2 – Tabela Datum_H

```
-- Table: public.datum_h
-- DROP TABLE public.datum_h;
CREATE TABLE public.datum_h
(
  id integer NOT NULL, -- Codigo numerico EDMO
  datum_h text, -- Descricao do sistema coord.
  ordem integer,
  CONSTRAINT datum_h_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.datum_h
  OWNER TO postgres;
COMMENT ON COLUMN public.datum_h.id IS 'Codigo numerico EDMO';
COMMENT ON COLUMN public.datum_h.datum_h IS 'Descricao do sistema coord.';
```

3.3 - Tabela Datum_V

```
-- Table: public.datum_v
-- DROP TABLE public.datum_v;
```

```
CREATE TABLE public.datum_v
(
  id text NOT NULL,
  datum_v text,
  ordem integer,
  CONSTRAINT datum_v_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.datum_v
  OWNER TO postgres;
```

3.4 - Tabela Entidade

```
-- Table: public.entidade
-- DROP TABLE public.entidade;
CREATE TABLE public.entidade
(
  id text NOT NULL, -- Codigo EDMO
  entidade text, -- Descricao da entidade
  CONSTRAINT entidade_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.entidade
  OWNER TO postgres;
COMMENT ON COLUMN public.entidade.id IS 'Codigo EDMO';
COMMENT ON COLUMN public.entidade.entidade IS 'Descricao da entidade';
```

3.5 – Tabela Equipamento

```
-- Table: public.equipamento
-- DROP TABLE public.equipamento;
CREATE TABLE public.equipamento
(
  id text NOT NULL, -- Codigo alfa numerico EDMO
  equipamento text, -- Descricao do equipamento
  CONSTRAINT equip_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
```

```
ALTER TABLE public.equipamento
  OWNER TO postgres;
COMMENT ON COLUMN public.equipamento.id IS 'Codigo alfa numerico EDMO';
COMMENT ON COLUMN public.equipamento.equipamento IS 'Descricao do equipamento';
```

3.6 – Tabela Parametro

```
-- Table: public.parametro
-- DROP TABLE public.parametro;
CREATE TABLE public.parametro
(
  id text NOT NULL,
  parametro text,
  CONSTRAINT pkey_parametro PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.parametro
  OWNER TO postgres;
```

3.7 – Tabela Plataforma

```
-- Table: public.plataforma
-- DROP TABLE public.plataforma;
CREATE TABLE public.plataforma
(
  id text NOT NULL, -- Codigo alfa numerico EDMO
  platform_class text, -- Descricao da plataforma
  CONSTRAINT plataforma_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.plataforma
  OWNER TO postgres;
COMMENT ON COLUMN public.plataforma.id IS 'Codigo alfa numerico EDMO';
COMMENT ON COLUMN public.plataforma.platform_class IS 'Descricao da plataforma';
```

3.8- Tabela Unid_Tempo

```
-- Table: public.unid_tempo
-- DROP TABLE public.unid_tempo;
CREATE TABLE public.unid_tempo
(
  id text NOT NULL,
  unid_tempo text,
  CONSTRAINT pkey_unid_tempo PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.unid_tempo
  OWNER TO postgres;
```

4 – Criação das tabelas principais

4.1 – Tabela Cdi

```
-- Table: public.cdi
-- DROP TABLE public.cdi;
CREATE TABLE public.cdi
(
  id integer NOT NULL DEFAULT nextval('cdi_id_seq'::regclass),
  codigo text,
  datum_h integer,
  datum_v text,
  min_depth real,
  max_depth real,
  area_type text,
  start_date timestamp with time zone,
  end_date timestamp with time zone,
  unid_tempo text,
  abstract text,
  platform_class text,
  holding_centre text,
  originator text,
  distributor text,
  collate_centre text,
  data_size integer,
  data_access text,
  cruise_name text,
```

```
  cruise_id text,
  qc_desc text,
  qc_date timestamp with time zone,
  qc_comment text,
  qc_status boolean,
  revision_date timestamp with time zone,
  long_w real,
  long_e real,
  lat_n real,
  lat_s real,

CONSTRAINT pkey_cdi PRIMARY KEY (id),
  CONSTRAINT "DatumH_fkey" FOREIGN KEY (datum_h)
      REFERENCES public.datum_h (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT "DatumV_fkey" FOREIGN KEY (datum_v)
      REFERENCES public.datum_v (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT acesso_fkey FOREIGN KEY (data_access)
      REFERENCES public.acesso (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT collate_center_fkey FOREIGN KEY (collate_centre)
      REFERENCES public.entidade (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT distributor_fkey FOREIGN KEY (distributor)
      REFERENCES public.entidade (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT holding_centre_fkey FOREIGN KEY (holding_centre)
      REFERENCES public.entidade (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT originator_fkey FOREIGN KEY (originator)
      REFERENCES public.entidade (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT plataforma_fkey FOREIGN KEY (platform_class)
      REFERENCES public.plataforma (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT unid_tempo_fkey FOREIGN KEY (unid_tempo)
      REFERENCES public.unid_tempo (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT unique_codigo UNIQUE (codigo)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.cdi
  OWNER TO postgres;

-- Index: public."fki_DatumH"
-- DROP INDEX public."fki_DatumH";
CREATE INDEX "fki_DatumH"
```

```
    ON public.cdi
    USING btree
    (datum_h);

  -- Index: public."fki_DatumV"
  -- DROP INDEX public."fki_DatumV";
  CREATE INDEX "fki_DatumV"
    ON public.cdi
    USING btree
    (datum_v COLLATE pg_catalog."default");

  -- Index: public.fki_acesso_fkey
  -- DROP INDEX public.fki_acesso_fkey;
  CREATE INDEX fki_acesso_fkey
    ON public.cdi
    USING btree
    (data_access COLLATE pg_catalog."default");

  -- Index: public.fki_collate_center_fkey
  -- DROP INDEX public.fki_collate_center_fkey;
  CREATE INDEX fki_collate_center_fkey
    ON public.cdi
    USING btree
    (collate_centre COLLATE pg_catalog."default");
  -- Index: public.fki_distributor_fkey
  -- DROP INDEX public.fki_distributor_fkey;
  CREATE INDEX fki_distributor_fkey
    ON public.cdi
    USING btree
    (distributor COLLATE pg_catalog."default");

  -- Index: public.fki_holding_centre_fkey
  -- DROP INDEX public.fki_holding_centre_fkey;
  CREATE INDEX fki_holding_centre_fkey
ON public.cdi
    USING btree
    (holding_centre COLLATE pg_catalog."default");

  -- Index: public.fki_originator_fkey
  -- DROP INDEX public.fki_originator_fkey;
  CREATE INDEX fki_originator_fkey
    ON public.cdi
    USING btree
    (originator COLLATE pg_catalog."default");

  -- Index: public.fki_plataforma_fkey
  -- DROP INDEX public.fki_plataforma_fkey;
  CREATE INDEX fki_plataforma_fkey
    ON public.cdi
    USING btree
```

```
  (platform_class COLLATE pg_catalog."default");

-- Index: public.fki_unid_tempo_fkey
-- DROP INDEX public.fki_unid_tempo_fkey;
CREATE INDEX fki_unid_tempo_fkey
  ON public.cdi
  USING btree
  (unid_tempo COLLATE pg_catalog."default");
```

4.2 - Tabela Patches

```
-- Table: public.patches
-- DROP TABLE public.patches;
CREATE TABLE public.patches
(
  id integer NOT NULL DEFAULT nextval('patches_id_seq'::regclass),
  cdi integer,
  pa pcpatch(3),
  CONSTRAINT patches_pkey PRIMARY KEY (id),
  CONSTRAINT patches_cdi_fkey FOREIGN KEY (cdi)
      REFERENCES public.cdi (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.patches
  OWNER TO postgres;
GRANT ALL ON TABLE public.patches TO postgres;
GRANT ALL ON TABLE public.patches TO public;
ALTER TABLE public.patches ALTER COLUMN pa SET STORAGE MAIN;

-- Index: public.fki_patches_cdi_fkey
-- DROP INDEX public.fki_patches_cdi_fkey;
CREATE INDEX fki_patches_cdi_fkey
  ON public.patches
  USING btree
  (cdi);

-- Index: public.indxgeom_patches
-- DROP INDEX public.indxgeom_patches;
CREATE INDEX indxgeom_patches
  ON public.patches
  USING gist
  (geometry(pa));
```

4.3 - Tabela Patches_Info

```
-- Table: public.patches_info
-- DROP TABLE public.patches_info;
CREATE TABLE public.patches_info
(
  id integer NOT NULL,
  codigo text,
  n bigint,
  prof real,
  geom geometry,
  prof_max double precision,
  prof_min double precision,
  CONSTRAINT pkey PRIMARY KEY (id),
  CONSTRAINT patches_info_cdi_fkey FOREIGN KEY (id)
      REFERENCES public.cdi (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.patches_info
  OWNER TO postgres;
```

4.4 - Tabela Patches_Sel

```
-- Table: public.patches_sel
-- DROP TABLE public.patches_sel;
CREATE TABLE public.patches_sel
(
  cdi integer,
  id integer,
  pa pcpatch(3),
  CONSTRAINT patches_sel_cdi_fkey FOREIGN KEY (cdi)
      REFERENCES public.cdi (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE,
  CONSTRAINT patches_sel_patches_fkey FOREIGN KEY (id)
      REFERENCES public.patches (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.patches_sel
```

```
  OWNER TO postgres;

-- Index: public.fki_patches_sel_cdi_fkey
-- DROP INDEX public.fki_patches_sel_cdi_fkey;
CREATE INDEX fki_patches_sel_cdi_fkey
  ON public.patches_sel
  USING btree
  (cdi);

-- Index: public.fki_patches_sel_patches_fkey
-- DROP INDEX public.fki_patches_sel_patches_fkey;
CREATE INDEX fki_patches_sel_patches_fkey
  ON public.patches_sel
  USING btree
  (id);

-- Index: public.indxgeom_patches_sel
-- DROP INDEX public.indxgeom_patches_sel;
CREATE INDEX indxgeom_patches_sel
  ON public.patches_sel
  USING gist
  (geometry(pa));
```

4.5 - Tabela Pontos_Temp

```
-- Table: public.pontos_temp
-- DROP TABLE public.pontos_temp;
CREATE TABLE public.pontos_temp
(
  x double precision,
  y double precision,
  z double precision,
  i integer,
  r smallint,
  g smallint,
  b smallint
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.pontos_temp
  OWNER TO postgres;
```

4.6 – Tabela Pc_Points

```
-- Table: public.pc_points
-- DROP TABLE public.pc_points;
CREATE TABLE public.pc_points
(
  id integer NOT NULL DEFAULT nextval('pc_points_id_seq'::regclass),
  pt pcpoint(3),
  CONSTRAINT pc_points_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.pc_points
  OWNER TO postgres;
GRANT ALL ON TABLE public.pc_points TO postgres;
GRANT ALL ON TABLE public.pc_points TO public;
```

5 – Criação das tabelas de ligação (implementação de relações do tipo "muitos para muitos")

5.1 – Tabela Cdi_Equip

```
-- Table: public.cdi_equip
-- DROP TABLE public.cdi_equip;
CREATE TABLE public.cdi_equip
(
  cdi integer NOT NULL,
  equip_id text NOT NULL,
  CONSTRAINT cdi_equip_cdi_fkey FOREIGN KEY (cdi)
      REFERENCES public.cdi (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE,
  CONSTRAINT cdi_equip_equipamento_fkey FOREIGN KEY (equip_id)
      REFERENCES public.equipamento (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE RESTRICT
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.cdi_equip
  OWNER TO postgres;

-- Index: public.fki_cdi_equip_equipamento_fkey
-- DROP INDEX public.fki_cdi_equip_equipamento_fkey;
CREATE INDEX fki_cdi_equip_equipamento_fkey
  ON public.cdi_equip
  USING btree
  (equip_id COLLATE pg_catalog."default");

-- Index: public.fki_cdi_fkey
-- DROP INDEX public.fki_cdi_fkey;
CREATE INDEX fki_cdi_fkey
  ON public.cdi_equip
  USING btree
  (cdi);
```

5.2 – Tabela Cdi_Param

```
-- Table: public.cdi_param
-- DROP TABLE public.cdi_param;
CREATE TABLE public.cdi_param
(
  cdi integer NOT NULL,
  param_id text NOT NULL,
  CONSTRAINT cdi_param_cdi_fkey FOREIGN KEY (cdi)
      REFERENCES public.cdi (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE,
  CONSTRAINT cdi_param_parametr FOREIGN KEY (param_id)
      REFERENCES public.parametro (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE RESTRICT
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.cdi_param
  OWNER TO postgres;

-- Index: public.fki_cdi_param_parametr
-- DROP INDEX public.fki_cdi_param_parametr;
CREATE INDEX fki_cdi_param_parametr
  ON public.cdi_param
  USING btree
  (param_id COLLATE pg_catalog."default");

-- Index: public.fki_param_cdi_fkey
-- DROP INDEX public.fki_param_cdi_fkey;
CREATE INDEX fki_param_cdi_fkey
  ON public.cdi_param
  USING btree
  (cdi);
```

6 – Definição de “Views” e funções

6.1 – “View” Vv_Patches

```
-- View: public.vv_patches
-- DROP VIEW public.vv_patches;
CREATE OR REPLACE VIEW public.vv_patches AS
 SELECT patches.cdi,
    patches.id,
    patches_info.codigo,
    geometry(patches.pa) AS geom,
    st_centroid(geometry(patches.pa)) AS pp,
    pc_patchavg(patches.pa, 'z'::text) AS prof,
    pc_numpoints(patches.pa) AS n
   FROM patches
     LEFT JOIN patches_info ON patches.cdi = patches_info.id;

ALTER TABLE public.vv_patches
  OWNER TO postgres;
```

6.2 – “View” Vv_Patches_Sel

```
-- View: public.vv_patches_sel

-- DROP VIEW public.vv_patches_sel;

CREATE OR REPLACE VIEW public.vv_patches_sel AS
 SELECT patches_sel.cdi,
    patches_sel.id,
    cdi.codigo,
    geometry(patches_sel.pa) AS geom,
    st_centroid(geometry(patches_sel.pa)) AS pp,
    pc_patchavg(patches_sel.pa, 'z'::text) AS prof,
    pc_numpoints(patches_sel.pa) AS n
   FROM patches_sel
     LEFT JOIN cdi ON patches_sel.cdi = cdi.id;

ALTER TABLE public.vv_patches_sel
  OWNER TO postgres;
```

6.3 – Função F_Pontos

```
-- Function: public.f_pontos(integer, integer)
-- DROP FUNCTION public.f_pontos(integer, integer);
CREATE OR REPLACE FUNCTION public.f_pontos(IN parm1 integer, IN parm2
integer)  RETURNS TABLE(n bigint, id integer, cdi integer, pt geometry, z numeric)
AS
$BODY$
SELECT  row_number() over() as n, temp.id,     temp.cdi,
st_makepoint(temp.x::double precision, temp.y::double precision) AS geom, temp.z AS
prof   FROM ( SELECT patches.id,  patches.cdi,  pc_get(pc_explode(patches.pa),
'x'::text) AS x,   pc_get(pc_explode(patches.pa), 'y'::text) AS y,
pc_get(pc_explode(patches.pa), 'z'::text) AS z   FROM patches
WHERE patches.cdi = $1 And id = $2)  temp;
$BODY$
  LANGUAGE sql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION public.f_pontos(integer, integer)
  OWNER TO postgres;
```

6.4 – Função F_PontosCdi

```
-- Function: public.f_pontoscdi(integer)
-- DROP FUNCTION public.f_pontoscdi(integer);
CREATE OR REPLACE FUNCTION public.f_pontoscdi(IN parm1 integer)
  RETURNS TABLE(n bigint, id integer, cdi integer, pt geometry, z numeric) AS
$BODY$
SELECT row_number() over() as n, temp.id,   temp.cdi,
st_SetSrid(st_makepoint(temp.x::double precision, temp.y::double precision), 4326) AS
geom,  temp.z AS prof
FROM ( SELECT patches.id,  patches.cdi, pc_get(pc_explode(patches.pa), 'x'::text) AS
x,  pc_get(pc_explode(patches.pa), 'y'::text) AS y,
pc_get(pc_explode(patches.pa), 'z'::text) AS z  FROM patches
WHERE patches.cdi = $1)  temp;
$BODY$
  LANGUAGE sql VOLATILE
  COST 100
  ROWS 1000;
ALTER FUNCTION public.f_pontoscdi(integer)
  OWNER TO postgres;
```