Copy No. _____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Implementation of the rapid environmental assessment production database

Michel Mayrand
OODA Technologies Inc.

Prepared by:
OODA Technologies Inc.
4891 Av. Grosvenor
Montreal, QC H3W 2M2

Project Manager: Dr. John Osler, 902-426-3100 ext. 119
Contract Number: W7707-098247/001/HAL
Contract Scientific Authority: Anthony W. Isenor, 902-426-3100 ext. 106

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2010-123
October 2010

Canada

This page intentionally left blank.

Implementation of the rapid environmental assessment production database

Michel Mayrand

Prepared by:

OODA Technologies Inc.
4891 Av. Grosvenor, Montreal Qc, H3W 2M2

Project Manager: Dr. John Osler 902-426-3100 Ext. 119
Contract Number: W7707-098247/001/HAL
Contract Scientific Authority: Anthony W. Isenor 902-426-3100 Ext. 106

Defence R&D Canada – Atlantic

Contract Report

DRDC Atlantic CR 2010-123

October 2010

Abstract

This contract deals with the construction of the redesigned Rapid Environmental Assessment (REA) database. The data model, as described in the report *Utilizing Arc Marine concepts for designing a geospatially enabled database to support rapid environmental assessment* [1] was implemented. SQL scripts were constructed for porting the data from the initial READBv3b to the new REA Production database. A user interface for executing these SQL scripts was also created. Functionality to validate the porting process was also included. A user-exit was created for accessing the ETOPO2 bathymetric data located in an external NetCDF file. This report contains the details of the porting process, all the instructions for using the user interface, the SQL scripts code, and finally a list of recommendations and possible tasks for future improvements.

Résumé

Ce contrat traite de la création du nouveau modèle de la base de données *Rapid Environmental Assessment (REA)*. Le modèle de données, tel que décrit dans le rapport *Utilizing Arc Marine concepts for designing a geospatially enabled database to support rapid environmental assessment* [1] fut réalisé. Des fichiers de commandes SQL furent construits pour migrer les données de l'ancienne base de données READBV3b à la nouvelle base de données REA dite de production. Une interface graphique fut créée pour contrôler l'exécution des fichiers de commandes SQL. La validation du processus de migration est assurée par un ensemble de fonctions SQL. Une fonction SQL spéciale (*user-exit*) fut construite pour accéder aux données bathymétriques ETOPO2 contenues dans un fichier externe en format NetCDF. Ce rapport décrit en détails le processus de migration des données, toutes les instructions pour l'utilisation de l'interface graphique, les fichiers de commandes SQL, ainsi qu'une liste de recommandations et de tâches potentielles pour de futures améliorations.

This page intentionally left blank.

Executive summary

Implementation of the rapid environmental assessment production database

Michel Mayrand; DRDC Atlantic CR 2010-123; Defence R&D Canada – Atlantic; October 2010.

Background: The REA database, which was first implemented in 2006, was primarily used for the storage of DRDC Atlantic maritime environmental data. As use of the REA database increased, it became desirable to redesign the database to better serve the user community within DRDC Atlantic. The redesign effort focused on the use of widely used standards and specifications for oceanographic data and metadata management. The redesign created a complete and highly documented data model including full documentation on the mapping of data from the existing REA database to the redesigned production data model.

Results: The porting of the data to the newly designed database was implemented under contract by OODA Technologies over a period of five months. This document describes the user interface for porting the data from the old database (READBv3b) to the new database. Functionality to validate the porting process was also included. A user-exit was created for accessing the ETOPO2 bathymetric data located in an external NetCDF file. This report contains the details of the porting process and the instructions for using the user interface.

Significance: This work contributes to the long-term management of environmental data at DRDC Atlantic. This is an important component to the continuing development of acoustic propagation models and the understanding of how environmental conditions impact the underwater acoustic environment. This understanding is critical to the progression of undersea warfare systems. As well, the development progresses the utilization of information management standards in applications to support naval research. Such standards are widely regarded as a key component of interoperability, and the utilization and understanding of these standards will contribute to CF efforts towards interoperability.

Future Plans: The redesigned database will be interfaced to acoustic models to assess the impact of environmental uncertainty on the acoustic predictions. Assessing the environmental uncertainty in this way, allows for directed environmental sampling. Research in this area will help guide the procedures developed for the rapid environmental assessment process.

Sommaire

Implementation of the rapid environmental assessment production database

Michel Mayrand ; DRDC Atlantic CR 2010-123 ; R & D pour la défense Canada – Atlantique ; octobre 2010.

Contexte : La base de données *REA*, réalisée en 2006, était principalement utilisée pour le stockage de données environnementales maritimes à RDDC Atlantique. Comme l'utilisation de la base de données *REA* augmenta avec le temps, une remodèlisation de la base de données s'imposait afin de mieux servir la communauté des chercheurs de RDDC Atlantique. L'accent de la remodèlisation fut mis sur l'utilisation des principales normes utilisées et spécifications des données océanographiques ainsi que la gestion des métadonnées. Le processus de remodèlisation a abouti à un modèle de données complet et très documenté incluant la description détaillée du processus de migration des données de l'ancienne base de données *REA* à la nouvelle base de données dite de production.

Résultats : La migration des données de *READBV3* vers le nouveau modèle de données fut implémentée par OODA Technologies sur une période de cinq mois. Ce document décrit l'interface graphique pour la migration des données de la base de données originale *READBV3* à la nouvelle base de données dite de production. Un ensemble de fonctions SQL fut crée pour valider le processus de migration. Une fonction SQL spéciale (*user-exit*) fut construite pour accéder aux données bathymétriques ETOPO2 contenues dans un fichier externe en format NetCDF. Ce rapport décrit en détails le processus de migration des données, ainsi que toutes les instructions pour l'utilisation de l'interface graphique.

Pertinence : Ce travail contribue à une meilleure gestion à long terme des données environnementales de RDDC Atlantique. Ceci est une composante importante pour la continuation du développement des modèles de propagation acoustique et notre compréhention de l'impact des conditions environnementales sur l'acoustique sous-marine. Cette compréhension est d'une importance critique pour l'évolution des systèmes de défense sous-marins. Aussi, ce développement contribue au progrès des normes de gestion de l'information dans les applications qui supportent la recheche navale. Ces normes sont reconnues comme une composante clé pour l'interopérabilité ; l'utilisation et la compréhension de ces normes contribura à l'effort des FC vers l'interopérabilité.

Plans futurs : La base de données remodelée sera interfacée avec les modèles acoustiques pour évaluer l'impact de incertitude environnementale sur les prédictions acoustiques. L'évaluation de l'incertitude environnementale de cette façon, permet un échantillonage dirigé de l'environnement. La recherche dans ce domaine contribura à raffiner les procédures développées pour l'évaluation rapide de l'environnement.

Table of contents

List of figures

1 Introduction

1.1 Scope

The objectives of the Rapid Environmental Assessment (REA) contract were to:

1. analyze and propose recommendations for the new Production data model (PDB) as described in [1]
2. create the PDB from the data model and populate it by porting data from an existing PostgreSQL database, as described in [2], to the PDB described in [1]
3. create metadata on the lineage of the ported data

The porting process in the request for proposal (RFP) was prioritized as follows:

1. expendable sound velocity (XSV) probe sound speed profile data
2. expendable bathythermograph (XBT) temperature profile data (if available)
3. sound speed profiles computed from XBT temperature data
4. DND maritime operation areas
5. Temperature-salinity Dalhousie (TSD) gridded data
6. sediment thickness gridded data
7. Non Acoustic Data Acquisition System (NADAS) data (i.e., includes along track air temperature, sea temperature, humidity, wind speed and direction, etc.)
8. Eastern Canada shallow (ECS) water ambient noise
9. bathymetry data (user exit)
10. sea bed forms (e.g., bedrock geology, bedrock outcrops, drift outcrops, faults, iceberg furrows, geoclutter, etc.
11. transmission loss data. Previously called the shallow water database and thus referred to as the SWDB.

The contract completed with all datasets, except the sea bed forms, as being ported from the load database (LDB) to the PDB. Discussions with the contract scientific authority in July 2010 identified priorities for the contract as concentrating on the quality of the data port, and resulting quality of data in the PDB. Specifically, the discussion highlighted the need for quality controlling the scripts used to produce the PDB, including comments within those scripts, investigation of a geometry problem within an area table, adding the filtering functionality to two data sets, and completing a quality final report.

This decision resulted in the sea bed form data port being dropped from the requirements. In part, this decision was based on the fact these data exist in shapefiles, and thus are already accessible by many GIS tools. As well, the decision resulted in lineage being a lower priority item as compared to the higher level goal of data quality.

Added to the completed list of data, is:

1. Scotian Shelf Sediment data

This data set was not identified in the original contract. The July 2010 re-prioritizing of effort resulted in this data set being added to the tasking.

1.2 Report outline

– In Section 2, we present the final data model with a list of the modifications implemented since the beginning of the project.
– In Section 3, experiences and decision rationale throughout the contract are presented.
– In Section 4, we described the SQL scripts for each package, underlining the lessons learnt and issues.
– In Section 5, we explain how to use the GUI interface for porting and verifying the database.
– In Section 6, we provide the sequential instructions for porting the data from the load database (LDB) using the command line.
– In Section 7, the instructions on how to install and execute the user-exit for the bathymetric data are presented.
– In Section 8, some design ideas and remaining questions on capturing the lineage data are listed.
– In Section 9, we summarize the recommendations for future improvement of the REA PDB.
– The codes for the scripts and user-exit program are presented in separate Annexes.

Throughout the document, opened questions, possible future tasks and recommendations appear in a framed box labelled has Q-#, T-# and R-#, respectively. This approach is proposed for facilitating the follow-up on those items. Possible task items also provide an estimate (in hours) for implementing the described feature.

2 Data model

In this section, we describe some of the conventions used in the PDB data model as well as the modifications made to the original model. For a detailed description of the original PDB data model, see Isenor and Spears [1].

The data model was produced with the Erwin data modelling software. Three versions of the data model were released during the contract:

1. At the contract kick-off (2010-02-02)
2. A month after (2010-03-01)
3. A frozen version, before the end of the contract (2010-05-28)

2.1 Modifications to the data model

The following conventions have been followed throughout the data model:

1. All table names are singular (except for the table named `series`, for which no singular exist)
2. If a table required a locally defined ID, it is written as the table name followed by the suffix `_ID` (e.g., Cruise_ID will be the ID for the table `Cruise`).
3. The use of the Camel Case convention (e.g., ShipName) is replaced by the underscore convention (e.g., `ship_name`). Most SQL implementations are case insensitive by default, case sensitivity is not always preserved between and inside/outside tools (e.g., Erwin and PostgreSQL). This is why the lowercase underscore convention is preferred to insure clarity and stability between tools. Some exceptions are acceptable such as automatic SQL code generation data modelling tools where case sensitivity can not be controlled by the user. Another advantage of the lowercase convention is when used with uppercase SQL keyword convention (e.g., `SELECT`, `INSERT`, etc.), it results in easier to read code.
4. The Camel Case convention is used only for SQL function names
5. The UTF8 encoding was chosen for the PDB. The LDB ASCII encoding was not even suitable for the original database as it can not support accentuated characters which can be potentially found in scientist and ship names, descriptions, notes, etc.

The following modifications were made to improve the original data model presented in [1]:

1. Table name `survey_info` was replaced by `survey`.
2. Table name `feature_assets` was replaced by `feature`.
3. The typing for field name `quality_flag_id` was changed from `CHAR(10)` to `INTEGER`.
4. In table name `transmission_loss`, the fields `bottom_type` and `sound_description` sizes are increased from `VARCHAR(20)` to `VARCHAR(200)` as the LDB values were larger than 20 characters.
5. In table name `transmission_loss`, a new column `original_station_id VARCHAR(20)` was added because the original LDB station id was alpha-numeric and not numeric only.
6. In table name `data`, field name `replicate_id` was replaced with `group_id`.

7. In table name `feature_area`, field name `feature_class` was replaced with `area_name`.
8. In table name `transmission_loss`, field name `band_width` was replaced with `bandwidth`.
9. In table name `mesh_point`, field name `point_type` was replaced with `mesh_point_type`.
10. The type of all `measuring_device_id` fields was changed from `INTEGER` to `BIGINT`.
11. In table name `quality_flag`, two new columns were added: `source VARCHAR(100)` and `source_flag VARCHAR(20)`.
12. For the ECS data, instead of using the suggested 1 to 4 for the `station_id`, a real unique ID was generated instead.

Figures 1, 2, and 3 show those parts of the data model pertinent to the data collected or used by DRDC Atlantic. Those sections of the data model dealing with lineage and metadata are not shown.

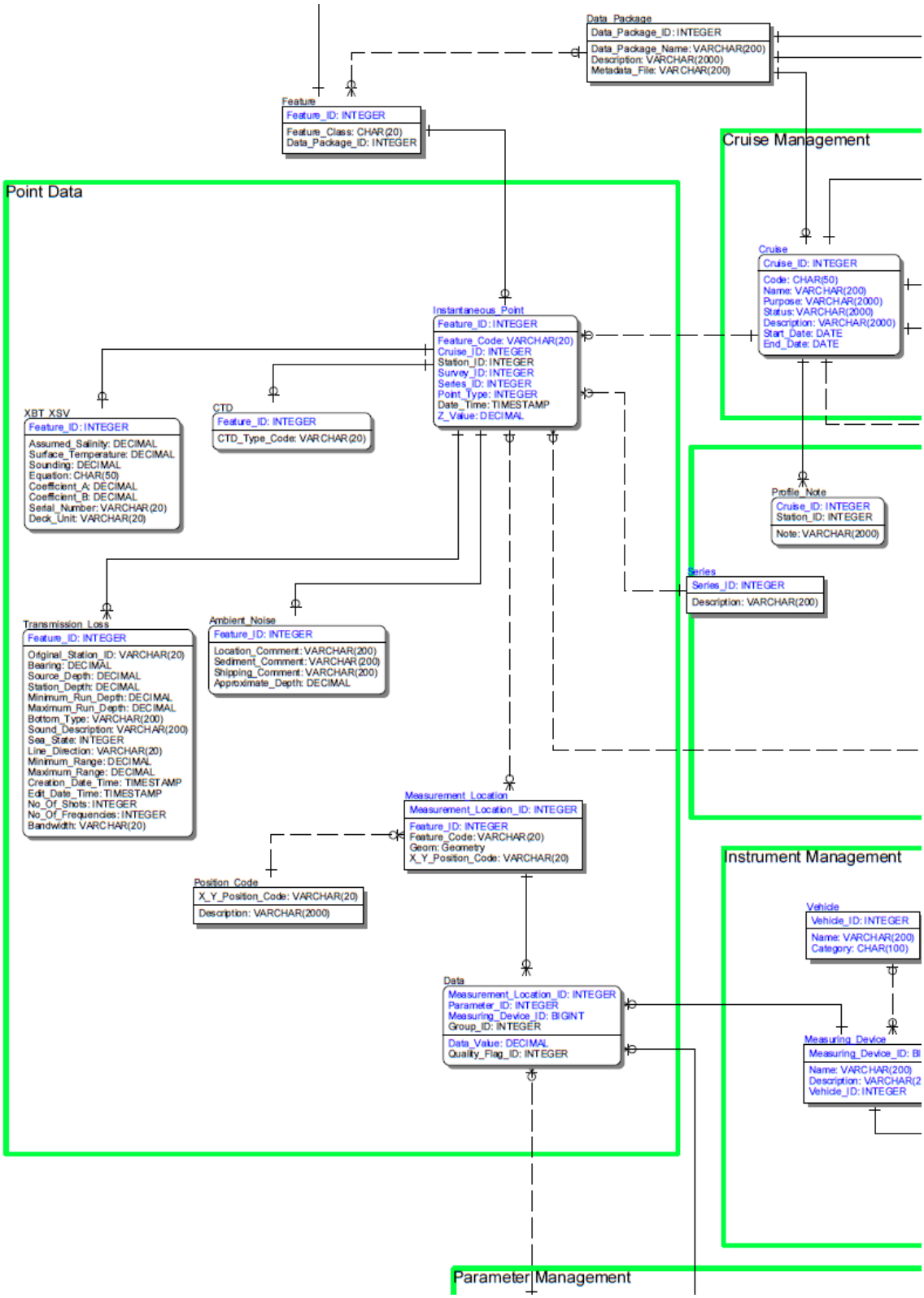

Figure 1: *Data model: Point Data.*

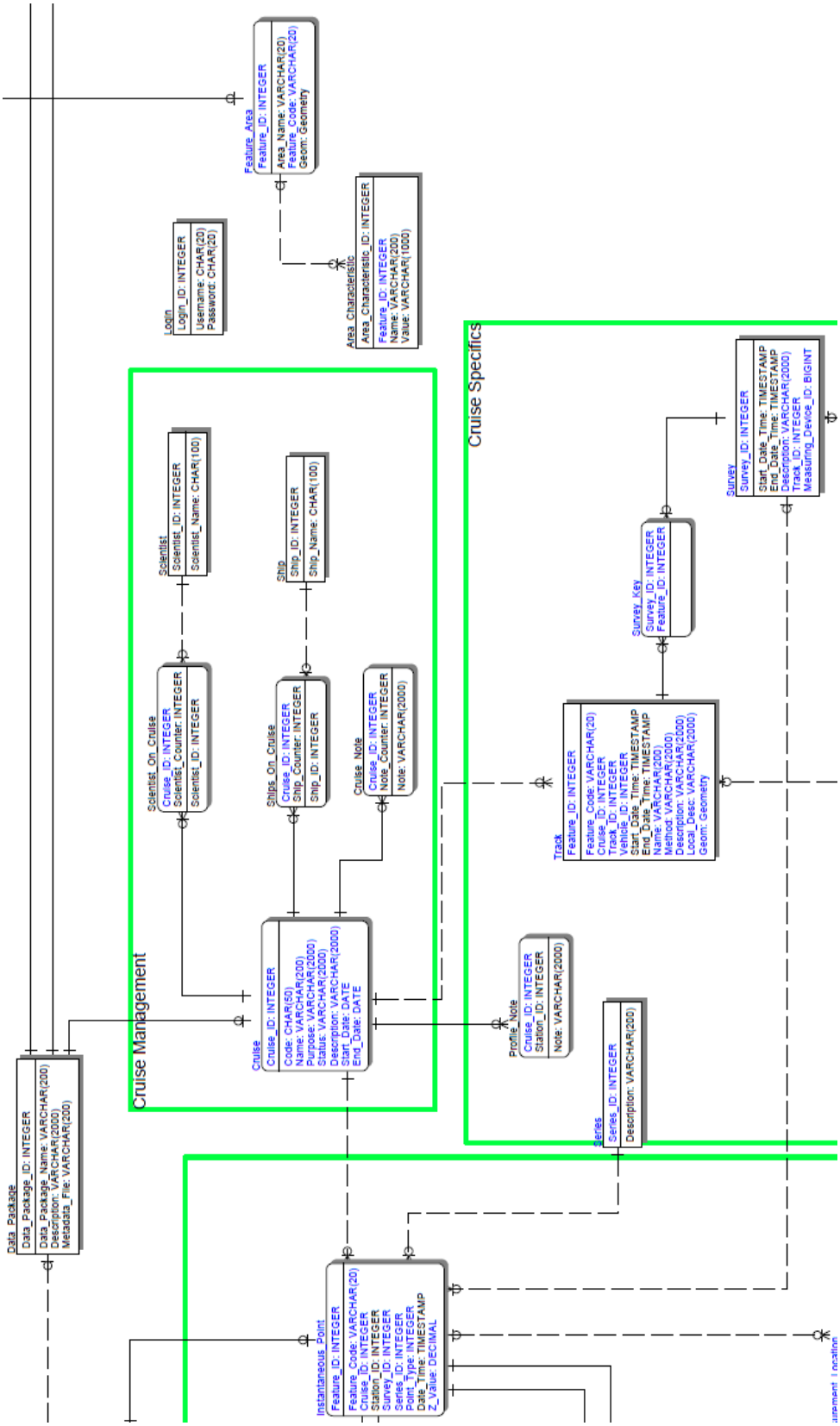

Figure 2: *Data model: Cruise Management, Cruise Specifics, Login, Feature Area related tables.*

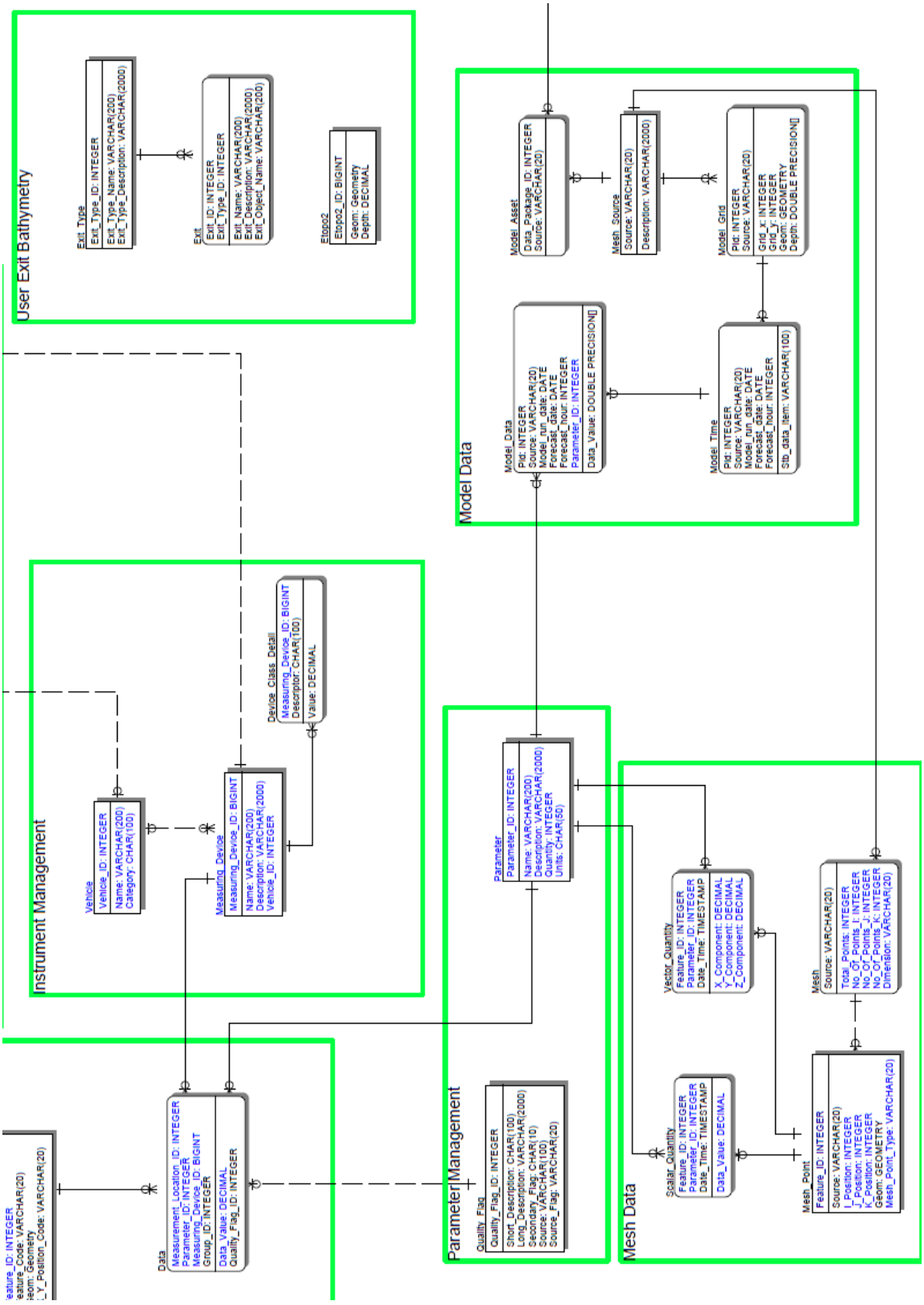

Figure 3: *Data model: Instrument Management, Parameter Management, Mesh Data, Model Data and user-exit Bathymetry tables.*

The PDB data model is a clear improvement to the original REA database. If one would like to improve the model further, a first possibility would to make it more symmetrical, between the Point Data and the Mesh Data. For instance, no quality flag is defined for the Mesh data, and there is a natural place holder for vectors in Mesh but not in Point Data. For example, instead of having Point and Mesh, one could have a *super-class* N-Dim Data which would hold the special case of Point (0-Dim) and Mesh (2-Dim), attributes (*measured*, *computed*) and (*regular* or *irregular*) could describe all the special cases. `Measurement_Location` could be merged with `Mesh_Point` and similarly for `Data` and `Scalar_Quantity`. Again, we could transform `Data` into a super-class that could hold scalar and vector, making the model even more symmetrical. Then again, this conceptual model may not hold when applied to real cases on a real database. However, a more symmetrical model would bring the data towards a single representation, making it easier for comparison and manipulation.

R1- The `group_ID` is still a patch for including multiple data (such as frequencies, transmission loss) with the same location. A better parameter should be found that can handle replicate data and arrays of data at the same location.
R2- To merge tables from Point and Mesh into a super-class N-Dim.

3 Lessons learnt and decision rationals

Before describing the scripts and GUI interface, it is useful to note the lessons learned during this project:

1. At the very beginning, the use of the PostgreSQL `dblink` module became mandatory in order to transfer *at the same time* data from one PostgreSQL database to another PostgreSQL database, even on the same computer and DBMS. The only other technique beside using the `dblink` module would have been to transfer data via a dump file, which would have been less streamlined and much slower. The use of `dblink` was not difficult to master and it was shown to be reliable for the purpose of the project.
2. During the course of the contract, many DB tools were tested. The three most useful ones were: `PgAdmin3`, `phpPgAdmin` and `pgsnap`. The first two are navigating/editing tools (specific to PostgreSQL). The third one (`pgsnap`) provides useful statistical and factual information on a PostgreSQL database. This tool was used to confirm that the load database used at the OODA office was identical to the load database supplied by DRDC Atlantic.
3. The amount of preprocessing needed for porting the data was underestimated by the contractor. Some of the processing needed was clearly stated in the Annex A of the report [1] but other important sources of preprocessing were only revealed when analyzing the Annex A with the other sections of the report. Also, other sources were found when analyzing the content of the LDB itself. Here are some examples:
   (a) Arrays to be ported required separation of array values into multiple scalar numbers.
   (b) Type text recast to numerical.
   (c) Timestamps created by fusing two separated values (date and time).
   (d) Units to be filtered out.
   (e) NULL values are filtered out
   (f) 3D geometry is formed from a 2D geometry fused with a depth value.
   (g) missing comma separator in arrays

   The amount of preprocessing also had consequences on the verification process. For example, how should the PDB field value that is a double precision number be compared to the LDB field value where it originated, when that double precision number was originally in the form of characters in the middle of an array that is defined as a string? Although not impossible, the verification process also required a large amount of preprocessing and would be slow because of the multiple steps needed for the comparison. Another technique would be for a given quantity, to create for each database a large array of that quantity (providing that the size and type are made compatible again) and to compare those two large arrays, where SQL is efficient in that kind of comparison. This technique can not detect the permutation of two quantities but would provide evidence that all the data was ported.
4. When asking about converting all geometry values to a single Spatial Reference System Identifier (i.e., SRID=4326), it was decided to wait until after the trial period in order to minimize the number of potential error sources. The conversion was implemented in the second phase.

5. Here are some of the errors encountered in the LDB that interfered with the porting process. A solution or a decision was applied to each of them.
   (a) In table `opareas`, `NOVERBER THREE` should be `NOVEMBER THREE`
   (b) In table `nadas_observation`, column `percipitation_mm` should be `precipitation_mm`
   (c) In TSD, there were only the month and the PDB required a timestamp.
   (d) In SWDB, the station ID is alpha-numerical but the PDB equivalent was numeric only.
   (e) There were no geometry data for 76477 NADAS reports (they were ignored).
   (f) Line 602 and 611 in the Annex A should have been set to "YES". The additional instruction that `bound = TRUE` for selecting the geometry was needed for line 611.
   (g) There were no rows with `bound = TRUE opareas ROMEO` (which was dismissed).
   (h) In table `swdb_tl_files`, row 4 was systematically missing the comma separator, constant lenght was assumed for separating the array values.
   (i) In SWDB, empty values for `hp_position` were assumed to be equal to zero.
   (j) In table `swdb_index`, 10 entries did not have a bearing. The data was ported without the bearing.
   (k) PDB type size for receiving columns Sound_Description and Bottom_Type were too small according to the LDB values. It was increased from varchar(20) to varchar(200).
   (l) There was no filename `Q0601A.DSWDB` (id: 2000031) in table `filenames`. We used the column `id_cruise` in `swdb_index` which was more complete.
   (m) `NULL` data values were found for XBT (sound speed) and for ECS data. These data were skipped.
   (n) In TSD data, there were complete arrays of zeros in `tsd_salinity` and `tsd_temperature`. These were dismissed.

R3- Surveys has its own tables and own unique survey IDs. Perhaps we should do the same for the station (with a unique station ID).
R4- For a large table, one should consider using `bigint` instead of `integer` for the ID. It is ok for now but as the database grows this could become a problem in the future. The candidates for `bigint` are `feature, data, measurement_location` and `instantaneous_point`.
R5- The `Device_Class_Detail` descriptor fields `XBTEquation, XBTCoefficientA, XBTCoeffientB` were not used. This was because there was no Isenor and Spears [1] Annex A instruction on this topic, in spite of [1] pg. 34 indicating that these fields should be used. This means the `Device_Class_Detail` table does not have information on the XBT equation or coefficients. However, the coefficients for each XBT profile are contained in the XBT_XSV table, so the historic processing coefficient values are retained in the PDB. The recommendation would be to clarify the exact purpose of the `Device_Class_Detail` usage of these descriptors.

Q1- No track nor survey key were generated during porting. Further instructions would be necessary on how to reconstruct the track and what defines a track start and end?
Q2- Similar question. No i,j,k positions were produced for mesh data. For the data sets present in the database, to determine the i,j,k points may require reverse engineering the geometry data. It is recognized that the i,j,k points are indices for the convenience of the user, and thus are not mandatory. However, the i,j,k points may make isolating single or groups of points easier for the users. One issue that may be encountered would be the procedure used to define the origin (the point 0,0,0). For the data sets loaded in the PDB, there was no unambiguous origin.

4 SQL scripts

4.1 Programming language

PL/PGSQL was selected first as the programming language for the scripts. This language is native to PostgreSQL and it has all the basic functionalities required in the porting process. Note that PL/PGSQL would not be applicable to the creation of user-exits. In the user-exit case, another language was chosen (see Section 7).

With the first SQL porting scripts, it became clear that because of the size of the databases and the amount of preprocessing needed, the scripts would take a non negligible time to run (hours). The choice to select another script language that could only run on top of SQL was less desirable as it would only provide overhead on the processing time.

4.2 Granularity

Different sizes of scripts were tested. Small scripts provided micro control over the data and modularity, but on the other hand, the total number of lines of code would have been greater than other scenarios. As well, the library of script files would become difficult to manage. Also, in that kind of granularity, we were loosing some performance advantages of the SQL language. This is because SQL was designed to efficiently execute functions on complete columns of data.

A single script file would have been out of the question. Although not impossible, it was not preferable. PL/PGSQL functions run under an implicit transaction principle (to protect the data integrity). This means if an error occurs in the transaction, all changes revert to their original state. So, a single error or interrupt could make hours of computation useless, forcing a re-start. Besides that, the monolithic script file would also be plagued with the consequences of non-modularity: low re-usability, interoperability, and maintenance difficulties.

4.3 Design

A natural equilibrium was found by using seventeen scripts regrouped as follow:

1. The original data model (Government furnished information (GFI), code not included in this document)
2. Sequences (see code in Annex A)
3. Alterations to the model (see code in Annex B)
4. Utility functions (see code in Annex C)
5. Login information (see code in Annex D)
6. Instrumentation management (see code in Annex E)
7. Parameter management (see code in Annex F)
8. Position code (see code in Annex G)
9. Cruise management (see code in Annex H)

10. XBT/XSV (see code in Annex I)
11. Areas (see code in Annex J)
12. TSD (see code in Annex K)
13. Sediment thickness (see code in Annex L)
14. NADAS (see code in Annex M)
15. ECSWAN (see code in Annex N)
16. SWDB (see code in Annex O)
17. Sediment SS (see code in Annex P)

It was advantageous and necessary to keep the datasets in separate scripts. A single dataset requires linking many different items (`feature_id`, `measurement_location_id`, `survey_id`, etc). Keeping the dataset in a single file prevents cross-over errors between datasets.

As a first iteration, the result is not yet optimal but promising. For example, we could easily include Position Code within the Parameter Management Script. Historically, they were not placed together as they were from different sections of the data model.

The first four scripts, namely `data_model.sql`, `sequences.sql`, `util_fct.sql` and `alterations.sql`, establish the foundation content that is used by the follow-on scripts. These follow-on scripts provide three primary functions:

1. insertAAA(): populate the PDB data with LDB data associated with the AAA dataset.
2. verifyAAA(): verify the PDB data with LDB data associated with the AAA dataset.
3. deleteAAA(): delete the PDB data associated with the AAA dataset. Note that this is a `delete`, not the same as `drop`, meaning that the rows are removed but the table structure remains.

These functions are executed at varying times, to perform the data porting (insert function), verifying the port, and the deletion of the ported data. Typically after the deletion, the insert function would be modified to correct identified errors.

This symmetrical design makes it easier for mapping a user interface to these functions. Also, maintenance and debugging of these functions is greatly simplified for such a symmetrical design.

In the following sub-section, we regroup the 17 scripts (the first one being the new data model itself) into two categories: Support Data (or non-georeferenced data) and Georeferenced data.

4.4 Support Data

In this section, we regrouped the scripts which need to be executed first as they are necessary for porting the other datasets.

4.4.1 Data model

The script Data Model is the first script to be executed. This script uses the original Erwin Data Model as input and creates the database schema as described in Section 2) and as shown in Figures 1, 2, and 3. The script creates each table with the corresponding columns and types. It also provides constraint instructions (`NOT NULL`, `DEFAULT` value, `CHECK` condition, etc) as well as `FOREIGN KEY` constraints. The SQL code for the data model was not reproduced in this document.

4.4.2 Sequences

This script (see Annex A) contains the sequences for the serial IDs used in some tables. The serial sequence provides the default ID when creating a new row. The serial sequence is not perfect, since row deletion will create *holes* in the ID sequence. But for porting the LDB to the PDB, this is not a problem as there are no rows being deleted in the process. Afterwards, one can change the sequence scheme to a more intelligent ID factory if needed. This would be particularly useful for dataset updates.

4.4.3 Utility functions

This script (see Annex B) contains some useful functions used in other scripts. The utility functions are:

1. **createFeature**: For a given data package ID, a function that creates a row in the feature table and returns the feature ID value to be referenced in other tables.
2. **getScientistNameFromID**: A function that provides the scientist name from the scientist ID number (LDB).
3. **getScientistIDFromName**: A function that provides the scientist ID (PDB) from the scientist name.
4. **getCruiseIDFromIDFilename**: A function that gives the Cruise ID associated to a LDB file ID.
5. **getMeasuringDeviceIDFromName**: A function that retrieves the measuring device ID in the PDB from a given device name.
6. **createTimestamp**: A function that merges a date with a time to make a timestamp.
7. **getCoordFromPointRangeBearing**: A function that gives the longitude and latitude of a point located at a certain range and bearing from a point of origin.
8. **triggerFeatureDelete**: A trigger function for deleting a Feature row upon deletion to other associated table (e.g., `instantaneous_point table`). This trigger has been deactivated for performance reasons.
9. **printTimeLeft**: A function approximating the time left for a script to terminate.

4.4.4 Alterations

This script (see Annex C) contains all the `ALTER TABLE` statements needed for the successful porting of the data. The majority of the `ALTER TABLE` statements are needed to assign sequence functions

and default values to the associated data model tables. The Erwin tool which generated the data model did not provide these sequence functions and this is why they were added afterward using `ALTER TABLE` statements. Some other `ALTER TABLE` statements are for lifting some constraints which are blocking the porting process. Without these statements, error messages would show up during the porting process and block the data inserts. As for the few constraints left, they are related to the PostGIS constraints such as the uniformity of the SRID and the dimensionality of the geometry. These `ALTER TABLE` statements were added at the beginning of the project because the original SRID and dimensions of the LDB triggered these constraints and prevented the migration process. Since then, the SRID from the LDB (SRID=4269; North America datum 1983 or NAD83) has been upgraded to the PDB SRID=4326 (World Geodetic System 1984, or WGS84). The associated `ALTER TABLE` statements could be now unnecessary and be removed after testing.

T1- To reactivate and retest the PostGIS constraint which was removed using ALTER TABLE command (3 hours).

4.4.5 Login

This script (see Annex D) transfers the login username and password from the LDB to the PDB. No preprocessing was needed in this script. The verification function for this script was fully implemented.

4.4.6 Instrumentation management

This script (see Annex E) is responsible for porting data to the `measuring_device` and `device_class_detail` table. The list is based on information found in different sections of the report [1]. Note that this table will be augmented later by the SWDB (`transmission_loss`) where rows will be added for each hydrophone (HP) of the mooring setup. No preprocessing was needed here. The verification script was fully implemented.

4.4.7 Parameter management

This script (see Annex F) is responsible for porting data to the parameter table. Again, the resulting list is based on information found in various sections of the report [1]. However, most of the content is from the `drdctype` table of the LDB. For this contract, the quality flag is assumed to be set to `unknown` for **all ported data**. Some numbers were chosen for custom-made parameter definitions. The numbers 401 and 402 were used for the package Sediment Thickness. The parameters ranging from 800 to 815 are used for the ECS and SWDB data. No preprocessing was needed here. The verification script was fully implemented.

4.4.8 Position code

This script (see Annex G) is responsible for porting data to the position code table. The list of data is based on information found in different sections of [1] but mostly from Annex D. Additional values

(HPDepth, HPNumber and HPPosition) were added in order to accommodate the Transmission Loss Table. No preprocessing was needed here. The verification script was fully implemented.

4.4.9 Cruise management

This script (see Annex H) is responsible for porting data to the Cruise Management section of the data model. The script will also automatically create a data_package row for each cruise ID listed in the original LDB. The data_package_id is also used in corresponding metadata tables when the inserted dataset is for XBT/XSV, NADAS and Transmission Loss. Processing was needed here in order to fill scientist_on_cruise because the ID used in the PDB is different from the ID used in the LDB. The verification script was fully implemented.

There was no source of information that linked a ship to a cruise. For porting the LDB, this was not a problem because there was only one ship in the database. But, if they were more than one ship, the information to link it to a cruise would have been missing.

For the purpose of this data port, the counter was set to 1, but in the case of feature updates, a better counter function, perhaps a trigger, should be designed to replace the default value.

Q3- The cruise table only contains the cruise ID. All the other columns are empties. Can we find information somewhere else to fill the other columns? Or do we leave it like this?

4.5 Georeferenced Datasets

The previous sections described scripts related to the porting of metadata. In this section, actual environmental data are ported using numerous data-specific scripts. All previous and supporting scripts must have been executed before executing any of the following scripts.

4.5.1 XBT/XSV

This script (see Annex I) transfers the LDB XBT/XSV data. The data_package_id was matched to the equivalent cruise_id. Many sources of preprocessing were needed for this package:

1. All the numerical metadata was defined as strings and needed to be cast to numerical values before porting.
2. Units embedded in the metadata had to be removed (e.g., assumed salinity, surface temperature)
3. The 2D geometry was changed to a 3D geometry using the depth as the Z value.
4. The timestamp was kept in a separate table geopoints and had to be matched and retrieved using the id_filename parameter.
5. The cruise_id needed in instantaneous_point was retrieved using the id_filename parameter.
6. NULL values for temperature and soundspeed were filtered out.

7. Preprocessing was needed also (detection of NULL temperature and NOT NULL soundspeed) to assign the proper measuring_device_id for the soundspeed data.
8. Computed values (as opposed to probe measurements) are also properly identified.

The x_y_position_code was set to unknown instead of GPS as we do not know when GPS was implemented in the XBT/XSV data. The verification script is 66% complete. Only the geometry data was missing in the verification procedure because of its 3D vs 2D status.

T2- To add a verification procedure to compare 3D geometry data with 2D (16 hours)

4.5.2 Areas

This script (see Annex J) transfers the information of the LDB opareas and marloa data. The data_package_id was set to 1000001 for OPAREA one and 1000002 for MARLOA. Information in [1] was insufficient to complete the port. Corresponding with the Scientific Authority, it was decided to only port rows from the original LDB table opareas_areas where the field *bound* = TRUE. This resulted on one row being dismissed. That row was for operation area ROMEO, which is associated with Halifax Harbour.

4.5.3 Temperature/Salinity Dalhousie dataset

This script (see Annex K) transfers the Temperature/Salinity Dalhousie dataset. The data_package_id is set to 3000001 for this package. Again, the information found in [1] was insufficient to complete the port. Specifically, there was no specific time stamp appropriate for the dataset. This meant the PDB tables for the mesh data were not appropriate; or that a time stamp needed to be created for the dataset. A constant year and date were assigned to each month in order to create the timestamp (2000-MM-01 00:00:00).

Many sources of preprocessing were needed for this package:

1. Temperature, salinity and depth were kept in arrays of length 15. By looping, we extracted the value in order to produce the scalar quantities with the corresponding Mesh Point.
2. The 2D geometry was changed to a 3D geometry using the depth as the Z value.
3. The month value was merged with a constant year, day of the month and time to produce a valid timestamp for the PDB table.
4. The geometry was located in a different table geopoints and had to be matched with the corresponding salinity and temperature value using an ID parameter.
5. 20% of the data were arrays of zeros (depth, salinity and temperature) corresponding to land area. All the zero data were filtered out.

With all the preprocessing, there was nothing left intact for a rapid comparison. Any attempt of verification will involve redoing the preprocessing steps and filtering in order to compare the values from both the LDB and PDB. The good news is that the salinity and temperature were stored as a real array of numbers and not strings to be parsed. The best technique for comparison would be to read the LDB arrays and to make a temporary table of scalar values, filtering out the zeros. Then, run a SELECT EXCEPT ALL SELECT kind of comparison between the temporary table and the

salinity data in the PDB. For the temperature, this would be slightly more difficult as there are valid zero temperature values to be kept.

T3- To improve the verification script to include the salinity, temperature and geometry data (8 hours)

4.5.4 Sediment thickness

This script (see Annex L) ports the geometry value and the sediment thickness into the `mesh_point` and `scalar_quantity` table, respectively. The `data_package_id` is set to 4000001 for this package. The only information missing in the LDB was the date. We used the value suggested in the report: July 24, 2006 at 15:59:00. Beside the setting of the timestamp, no preprocessing was needed here. The verification script was fully implemented.

Q4- How to interpret the multiple -9999 (associated to thickness) found in the sedthick table at the beginning?

4.5.5 NADAS

This script (see Annex M) transfers the NADAS data to the PDB. It is a big but fairly standard transfer. The `data_package_id` was matched to the equivalent `cruise_id`. As for preprocessing, some steps were needed:

1. The `cruise_id` needed in `instantaneous_point` was retrieved using the `id_filename` parameter.
2. `NULL` data values were filtered out.
3. The `x_y_position_code` was extracted from the processing of the (numerical) `the_geom_drdccode`. The value in this field was not documented in [1], but corresponds to a `drdccode` indicating the type of positioning used to determine the latitude and longitude. The integer value in the field was mapped to the `drdctype` in order to find its string equivalent and finally mapped to the closest match in the `x_y_position_code`.

It was surprising to find out that the NADAS script is by far the longest to run, by more than a factor of 10. Although the numbers of rows is half of the Sediment Thickness, the cause of time spent for this script is probably related to a very large number of `INSERT` statements (16). A performance analysis done on a small subset of the NADAS data using the PostgreSQL `EXPLAIN ANALYSE` debugging tool should shed some light on the problem. The verification script was fully implemented.

T4- Analyze the performance of the NADAS script using the SQL command `EXPLAIN ANALYSE` on a subset of the data to identify the sources of the CPU intensive subtasks. Based on the results, implement identified improvements. (8 hours)

4.5.6 Eastern Canadian Shallow Water Ambient Noise (ECSWAN) dataset

This script (see Annex N) transfers the ECSWAN dataset to the PDB. The `data_package_id` is set to 2000001 for this package. Many sources of preprocessing were needed for this package:

1. The 2D geometry associated to a location was changed into 3 different 3D geometries located on the apexes of an equilateral triangle with side length of 10 nautical miles, centered on the 2D location point, with one of the apex pointing true North. A depth of `30 m` was assumed throughout the package (assumption from [1], see Annex A, record 254). The new coordinates were calculated using the function `getCoordFromPointRangeBearing`, using `bearing = 0,120,240` and `range =` $\frac{10}{\sqrt{3}}$ NM, which is the distance between any of the apexes and the center of the triangle. `series_id` were use to uniquely define each apex of the triangle. For calculating the apexes position, the circumference of the earth was assumed to be $360 * 60$ nautical miles.
2. There is no `cruise_id` associated to this package, only flight ID, which is mapped into `survey_id`.
3. Some remapping of the ambient noise data into arrays was necessary to align the data and facilitate the porting process.
4. `NULL` values were filtered out.

The metadata are repeated at each apex of the triangle.

T5- To improve the verification script to include the geometry, the ambient noise mean and standard deviation (`parameter_id = 801, 802`) (16 hours)

R6- The metadata is repeated at each apex of the triangle. The design for this dataset should be reviewed if it could be improved somehow.

4.5.7 Shallow Water Database (Transmission loss dataset)

This script (see Annex O) transfers the SWDB (Transmission Loss) dataset. This is the most complex dataset to process because of the complexity of the experimental setup and protocol and because of the amount of metadata. The `data_package_id` was matched to the equivalent `cruise_id`. Many sources of preprocessing were needed for this package:

1. All the numerical metadata was defined as strings and needed to be cast to numerical values before porting.
2. The 2D geometry was changed to a 3D geometry using the `source_depth` as the Z value.
3. The parameters of each hydrophone (depth, position and number) were encoded in the `measuring_device_id` in a unique way to preserve its characteristics. The encoding was as follows: `1AAAABBBBCCCC` where `AAAA` is the number, `BBBB` is the depth and `CCCC` is the horizontal position of the hydrophone. All three values are fixed length with leading zeros if needed. Since there are three components for each hydrophone, on each mooring, the result is a considerable number of rows specifically for hydrophone components.

4. The `cruise_id` needed in `instantaneous_point` was retrieved using the `id_filename` parameter.
5. Many different kinds of arrays stored as strings needed to be parsed and recast into numerical values.
6. Special parsing was necessary for `fileline = 4` in table `swdb_tl_files`, because line 4 was systematically missing the comma separator, constant length was assumed for separating the array values.
7. The original station ID was alpha-numeric and not only integer as required by the PDB `instantaneous_point` table. Another column was created to keep the original station ID information, numerical part was parsed and ported as the new `station_id`.
8. Timestamps were produced by fusing `date` and `time` values, using the utility function `createTimestamp(date,time)`.
9. `NULL` values were filtered out.
10. During the port, it was identified that 10 rows from the LDB had missing bearing data. It was decided to port the rest of the data (range, frequency, transmission loss) without the bearing data.
11. The `group_id` was used to resolve the problem of having multiple instances of the same `parameter_id` on the same location (e.g., frequency/transmission loss data).

Q5- The mooring is placed along the ocean bottom (see Figure 10 in Isenor and Spears [1]). Is the bearing of the horizontal base (i.e., the direction the mooring is placed along the ocean bottom) relevant?

T6- To improve the verification script to include preprocessed Transmission Data (24 hours).

4.5.8 Scotian Shelf sediment

This script (see Annex P) ports the geometry value and seven parameters labeled from 803 to 809 into the `measurement_location` and `data` table, respectively. The only information missing in the LDB was the date. We used the value July 01, 1986 at 00:00:00 (the date of the initial publication was 1986). Beside the setting of the timestamp, no preprocessing was needed here. The `data_package_id` is set to 5000001 for this package. The verification script was fully implemented.

5 GUI

In this section we describe how to install and use the graphical user interface for porting the REA database to the PDB. This is a simple interface based on Swing Java Graphical Library. It was designed and implemented using a graphical layout builder tool called `Gola` (`http://gola.mathnium.com/`, which makes use of the `Pagelayout` package (`http://pagelayout.sourceforge.net/`). The Java code for the REA GUI was generated automatically using `Gola` and then modified manually for connecting the parts and adding actions to events.

The GUI was designed to facilitate the porting process, to minimize SQL programming errors, but also to provide a centralized control where the user can port sub-parts of the database, as requested in the contract. The GUI was designed and implemented rapidly with very little impact on other tasks and was designed to support, in a later release, the capturing of lineage data.

5.1 Installing and running the GUI

All files and libraries needed for the GUI can be found in the `gui` directory. To compile the GUI Java source code (adjust the classpath below to your installation settings):

```
javac -cp .:/install_dir/gui:/install_dir/gui/pagelayout1.16.jar Rea.java
```

To run the GUI:

```
java -cp .:/install_dir/gui:/install_dir/gui/pagelayout1.16.jar Rea
```

Figure 4 shows the interface.

5.2 GUI user guide

The GUI is made up of two tabs, the REA Porting tab and the Lineage tab (to be implemented later). The REA Porting tab displays four sections:

1. Top section: Input text field where the user defines the location of the porting scripts (e.g., `/home/mmayrand/script_sql`)
2. Middle section: Action selection section where the user can control which script and what action will be activated.
3. Bottom left section: Output area where the script results are displayed.
4. Bottom right section: Control section where the buttons for executing/interrupting actions are located.

As stated before, the porting process has been grouped into 17 script files, the first four (data model, sequences, alterations and utility functions) are included in the item denoted `Load Model & Fcts` in the middle section of the GUI window. The remaining 13 scripts are divided into a left and

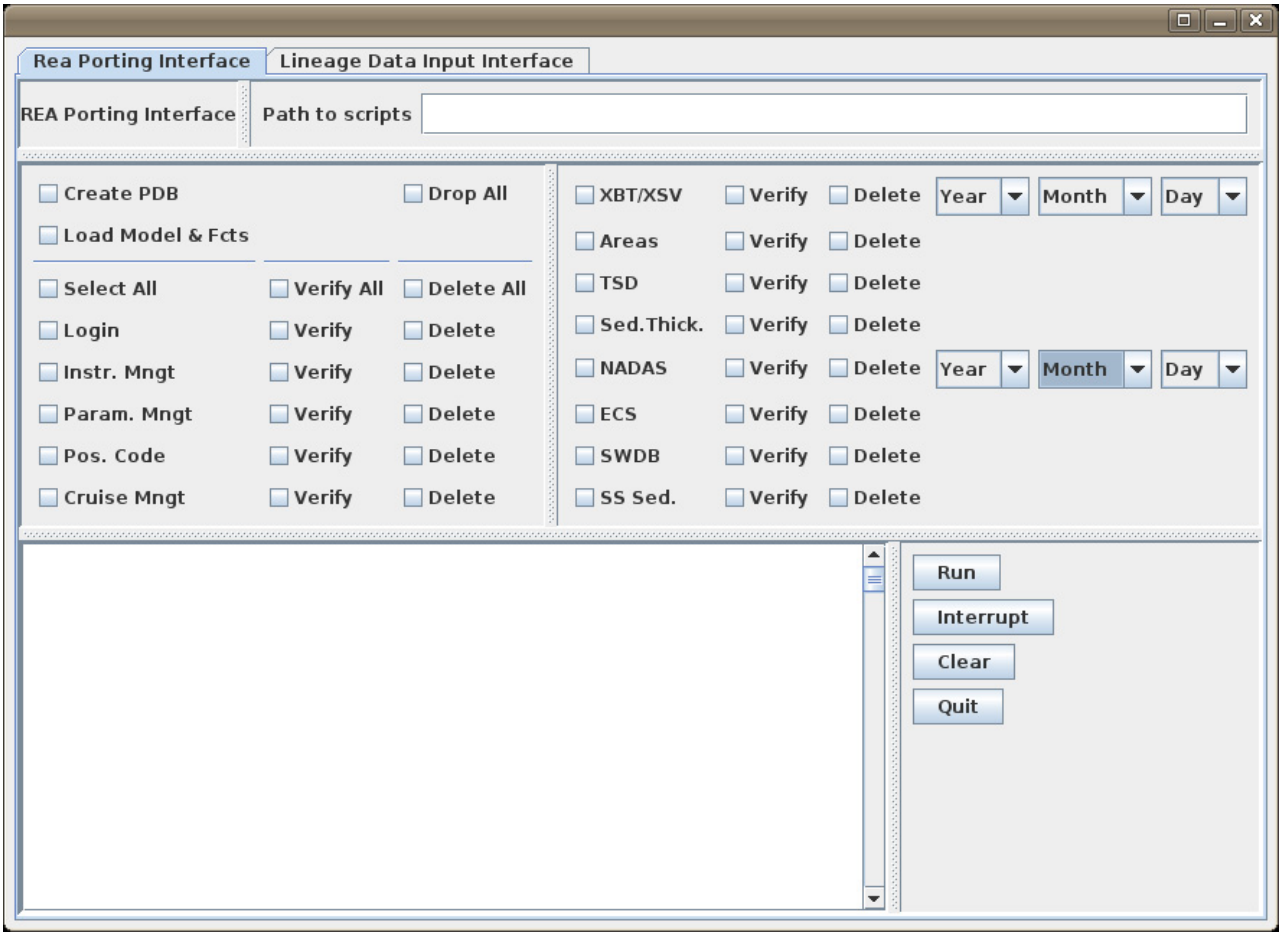

Figure 4: *User Interface for porting the REA database*

right panel. The left panel regroups all the support data needed for the packages of the right panel, which are the main datasets. The left and right panels are each subdivided in three columns, each representing one of the primary script functions: insertAAA(), verifyAAA() and deleteAAA().

The principle of the interface is fairly easy, by clicking a check box and then clicking on the button `run`, the user will execute the corresponding function. Important facts to know about the GUI:

1. The GUI is not sequential and can launch (because of threads) multiple selections in parallel. Some features were added to force sequential behaviour but it is not working properly yet. Because some scripts must be executed in a certain order, it is **highly recommended to run one check box at the time**.
2. It is assumed that porting the LDB to the PDB is performed on a computer running PostgreSQL 8.3 (or close) with PostGIS extension installed. Also, the language PL/PGSQL should be declared for the PDB database. The command `SELECT * FROM pg_language;` will display the languages defined for the DB you are in. The language can be declared with the system command `createlang -U postgres pdb plpgsql`.

3. Before porting the database, you must drop the old PDB (if it exists) using the `Drop All` button. A confirmation popup window will ask for confirmation. Then, you need to create a new empty PDB with the `Create PDB` button. Only then, may you proceed with the porting process.
4. The sequence for porting goes as follows:
   (a) dropping the old PDB database (if it exists)
   (b) creating a new empty PDB
   (c) the first script to execute is `Load Model & Fcts`
   (d) all the remaining items of the left side (first column): `Login, Instr. Mngt, Param. Mngt, Pos. Code, Cruise Mngt`. Run these scripts one at a time. These constitute the support data necessary for porting the other datasets that are present in the right panel.
   (e) After all the data from the left side have been loaded, only then may you start porting items from the right side (first column: `XBT/XSV, Areas, TSD, Sed. Thick, NADAS, ECS, SWDB, SS Sed`.). Again, run these one at the time. The order is not important at this stage.
   (f) Two datasets (XBT/XSB and NADAS) can be filtered using a user defined date field. After selecting the year, month and day, only entries subsequent to this date will be processed by the porting tool. The three parameters must be set to be effective. If not, or if no date is set, the complete dataset will be processed. This filtering feature was only set for the two datasets. It is the responsibility of the user to manually port items that could be needed for these two datasets such as new cruise data.
5. Half of the scripts are executed very quickly, others may take from 1 to 2 hours. The longest duration is the NADAS script which could take up to 16 hours to execute. The output display shows porting progress reports with the remaining time before completion.
6. After being ported, it is recommended to run the corresponding `Verify` script. The results of the `Verify` script will indicate either *OK or mismatch* in the output window.
7. The `Drop All` check box in the upper left section will delete (drop) the complete PDB. A confirmation popup window will ask for confirmation.
8. The `Delete` check boxes will indicate to run the corresponding delete function for this package. The delete function will remove the rows of the corresponding package without dropping the table (no need to reload the Model). Somehow, the time for deletion grows linearly if not exponentially with the size of the package. Instead of deleting rows of large packages, it is recommended to drop the database completely and reload the data through the scripts from the start until the deletion problem is understood and solved.
9. The `Interrupt` button was designed to send a cancel signal to the process in progress. When a process is canceled before termination, because of the *transactional* nature of PL/PGSQL, none of the previous operations should have been implemented, protecting data from corruption. Nevertheless, that feature should be used with caution.
10. If the `Verify` scripts show a seemingly endless list of errors, it is a sign that something has corrupted your PDB implementation. It is recommended to drop the database and start over.

11. The `Clear` button will remove any text in the output windows and reset all check boxes to *unchecked*.
12. After each run, all check boxes are reset to *unchecked* to prevent repeating the execution of insert scripts more than one time and potentially corrupting the PDB.

T7- Completing some of the GUI features (sequential processing) and adding new ones (dump to file option, user-exit, ETOPO2 file location, etc)(40 hours).

6 Command line execution of the scripts

It is not mandatory to use the GUI. All scripts activated via the GUI can also be activated via command line. This section describes the command line execution of the scripts. The sequence order needed for some of the scripts is shown in the list below. See the above GUI section for important facts about the scripts before running them.

```
dropdb -U postgres pdb
createdb -U postgres -E UTF8 pdb
psql -U postgres pdb < data_model.sql
psql -U postgres pdb < sequences.sql
psql -U postgres pdb < alterations.sql
psql -U postgres pdb < util_fct.sql
psql -U postgres pdb < login_fct.sql
psql -U postgres pdb < instrument_mngt_fct.sql
psql -U postgres pdb < parameter_mgnt_fct.sql
psql -U postgres pdb < position_code_fct.sql
psql -U postgres pdb < cruise_mngt_fct.sql
psql -U postgres pdb < areas_fct.sql
psql -U postgres pdb < ecs_fct.sql
psql -U postgres pdb < nadas_fct.sql
psql -U postgres pdb < sediment_ss_fct.sql
psql -U postgres pdb < sedthick_fct.sql
psql -U postgres pdb < swdb_fct.sql
psql -U postgres pdb < tsd_fct.sql
psql -U postgres pdb < xbtxsv_fct.sql
psql -U postgres pdb -c "SELECT insertLogin();"
psql -U postgres pdb -c "SELECT insertInstrumentManagement();"
psql -U postgres pdb -c "SELECT insertParameterManagement();"
psql -U postgres pdb -c "SELECT insertPositionCode();"
psql -U postgres pdb -c "SELECT insertCruiseManagement();"
psql -U postgres pdb -c "SELECT insertXBTXSV('2008','09','01');"
psql -U postgres pdb -c "SELECT insertAreas();"
psql -U postgres pdb -c "SELECT insertTSD();"
psql -U postgres pdb -c "SELECT insertSedThick();"
psql -U postgres pdb -c "SELECT insertNADAS('2008','09','01');"
psql -U postgres pdb -c "SELECT insertECS();"
psql -U postgres pdb -c "SELECT insertSWDB();"
psql -U postgres pdb -c "SELECT insertSedimentSS();"
pg_dump -U postgres pdb > pdb.sql
```

The above date input parameter for `insertXBTXSV` and for `insertNADAS` are given as examples. Note that the year must have 4 digits and the month and day field must have 2 digits, filling with a `0`

if necesssary. To run these functions without the date filter, one only needs to provide empty input fields (e.g., `insertXBTXSV('','','')`). The last statement creates a dump file of the resulting database. The resulting size is around 4.4GB (uncompressed). Until the sequential problem of the GUI is resolved, the command line execution should be preferred for overnight batch execution. Within the `scripts_sql` directory, an executable command line script `create_pdb` contains all the above commands. The user can modify the file to fit its installation setup. The command line can be launched for overnight execution without having to supervise it. The GUI would be useful for users who want to monitor closely the porting process and executing verification scripts at each step of the installation.

The resulting output PDB can be loaded with the commands:

```
dropdb -U postgres pdb
createdb -U postgres -E UTF8 pdb
psql -U postgres pdb < pdb.sql
```

Note that the PDB creation procedure does not include the user-exit (see section 7) which has to be edited first and installed separately.

This is the sequence to verify the proper porting of the data:

```
psql -U postgres pdb -c "SELECT verifyLogin();"
psql -U postgres pdb -c "SELECT verifyInstrumentManagement();"
psql -U postgres pdb -c "SELECT verifyParameterManagement();"
psql -U postgres pdb -c "SELECT verifyPositionCode();"
psql -U postgres pdb -c "SELECT verifyCruiseManagement();"
psql -U postgres pdb -c "SELECT verifyXBTXSV();"
psql -U postgres pdb -c "SELECT verifyAreas();"
psql -U postgres pdb -c "SELECT verifyTSD();"
psql -U postgres pdb -c "SELECT verifySedThick();"
psql -U postgres pdb -c "SELECT verifyNADAS();"
psql -U postgres pdb -c "SELECT verifyECS();"
psql -U postgres pdb -c "SELECT verifySWDB();"
psql -U postgres pdb -c "SELECT verifySedimentSS();"
```

When the script is successful, an `OK` is returned. If not, a `Mismatch` is returned with the associated parameter ID.

This is the sequence to delete (without dropping the database schema) the rows of the PDB:

```
psql -U postgres pdb -c "SELECT deleteLogin();"
psql -U postgres pdb -c "SELECT deleteInstrumentManagement();"
psql -U postgres pdb -c "SELECT deleteParameterManagement();"
```

```
psql -U postgres pdb -c "SELECT deletePositionCode();"
psql -U postgres pdb -c "SELECT deleteCruiseManagement();"
psql -U postgres pdb -c "SELECT deleteXBTXSV();"
psql -U postgres pdb -c "SELECT deleteAreas();"
psql -U postgres pdb -c "SELECT deleteTSD();"
psql -U postgres pdb -c "SELECT deleteSedThick();"
psql -U postgres pdb -c "SELECT deleteNADAS();"
psql -U postgres pdb -c "SELECT deleteECS();"
psql -U postgres pdb -c "SELECT deleteSWDB();"
psql -U postgres pdb -c "SELECT deleteSedimentSS();"
```

IMPORTANT: Note that there is a performance issue when deleting large datasets. For large tables, the deletion of the `measurement_location` required a large amount of time. In a similar way, using a DBMS trigger to remove rows can also require a large amount of time. This, however, was easy to disable. Dropping the database and starting over is much faster.

7 User-exit for bathymetric data

This section describes the design and installation steps for implementing a user-exit for the bathymetric data. The term *user-exit* means that the data of interest is not located in the PDB, but rather exists outside of the DBMS environment. For the particular case of the bathymetry implementation developed here, the data exist in a NetCDF formatted file located in the file system. The goal of the user-exit is to make a portion of the external data visible inside the PDB as a set of temporary tables which will be automatically deleted at the end of a session. The advantages of the user-exit are:

1. more manageable size for the PDB
2. an extremely simple upgrade mechanism for the large external dataset. **This simple upgrade is dependent on no changes in the format or structure of the external data set**. When no format changes have been made, the upgrade consists of simply replacing the old external data set file with the new file.

The disadvantages are:

1. the addition of heterogeneous mechanisms for enabling communication between the DBMS and the operating system. This is because different external data sets will require different communication mechanisms. The result will likely be unique communications for each external data set, which could result in communication management issues for the database administrator.
2. no feedback nor confirmation when data cross the border between the two environments. The user can only assess the success or failure of the transaction between external data set and the database, by examining the result provided in the database.
3. a less secure environment. This is because the user is now capable of executing programs outside of the DB environment.

Despite the disadvantages, the gains from this technique are still interesting, also the disadvantages are minimized by the fact that the user-exit mechanism will be used only once or twice during a given REA session.

7.1 Programming languages

For a user-exit, there are two choices that must be made with respect to programming languages:
– (A) the programming language for the external program itself. This program is responsible for retrieving the data from the external data set.
– (B) the programming language used within PostgreSQL to launch the external program.
For the first case, the choice of a programming language was directly related to the availability of a NetCDF supporting library. In this case, we have a large choice of languages: C, Fortran, C++, Java, MATLAB, Objective-C, Perl, Python, R, Ruby, Tcl/Tk. Java was selected for portability reasons (choice (A) above). Also, it did not require the installation of other compiler/interpreter packages as the Java compiler is a standard installation in Linux, Solaris and Windows environments. A second option would have been NetCDF-Perl. However, this package is less mature than NetCDF-Java and also, performance problems (with large files) could have been another issue in this project.

The programming language native to PostgreSQL is PL/PGSQL. PL/PGSQL can not be used for creating a user-exit in this particular case because in order to reach *external* of the database *roam* (in the Linux file system for example), an *untrusted* language is necessary. Here, the term *trusted/untrusted* is relative to the database itself. In the language of PostgreSQL, *untrusted* refers to a language that can control more than what is contained in the DBMS. Thus, *trusted* refers to a language that can only control what is inside the DBMS. Also, in the case of an *untrusted* language, processes launched outside the DBMS are not identified or owned by the DB user. In this case, there is no mapping between DB users and system users.

PL/PGSQL is by design, a *trusted* language (because it can only manipulate what exists inside the DBMS). Since there is no *untrusted* version of PL/PGSQL, to run an external program one has to rely on other PostgreSQL languages such as PL/PerlU or PL/JavaU (the trailing U indicates the untrusted version of trusted languages PL/Perl and Pl/Java, respectively). PL/PerlU and PL/JavaU were both expected to be efficient for the purpose of the user-exit implementation, but some experimentations were in favor of the PL/PerlU package:

1. Unlike PL/PerlU, PL/JavaU programs can not be run in stand-alone in the PostgreSQL environment, because this requires a parallel class to be implemented outside the DBMS. Although this feature was interesting as it would have provided more control, it was unfortunately more difficult to implement as compared to its counterpart PL/PerlU.
2. With PL/PerlU, there was no `classpath` to set up, making installation and execution simpler.
3. The installation of both PL/JavaU and PL/PerlU was easy and quick using the Synaptic Package Tool for the Linux Ubuntu environment. The command line `apt-get install postgresql-plperl-8.3` is also trivial. A manual installation of the Pl/JavaU was attempted in order to evaluate the installation process. The result of that evaluation was that the installation guide was not clear. As well, bugs were encountered and had to be solved manually. Linux installation tools are highly recommended in this case.
4. It must be emphasized that both PL/PerlU and PL/JavaU are *experimental* packages with small communities of developers. New functionalities and improvements are implemented at a slow rate. That being said, PL/PerlU has slightly more features than PL/JavaU.

As a result, PL/PerlU was selected for creating the launching program in PostgreSQL (decision (B) above).

7.2 User-exit design

An initial attempt was made at the user-exit implementation by using PL/PerlU to retrieve the external NetCDF file. Then, using the Perl module for NetCDF, to perform the data extraction and generate the ETOPO2 database table directly, all within PostgreSQL. The advantage of this approach would have been to avoid the necessity of using an external program (Java in this case) for processing the file and creating the database table. Unfortunately, the experimentation with this approach were very quickly plagued by many problems: the absence of some important functionalities in the NetCDF-Perl packages; the complex installation procedure; and debugging of those packages; and extremely poor performance and problems when dealing with large files. This design approach was then dropped.

The final design used for the user-exit was as follows:

1. A function written in PL/PerlU, will be executed by the user within the PostgreSQL environment. The function accepts four input arguments: the latitude, longitude of the upper left corner and the latitude, longitude of the lower right corner of a user-defined rectangle. The hemispheres are identified using $+$ or $-$. North and East are denoted using $+$; while South and West are denoted using $-$. This function launches an external Java program using those four arguments.
2. The Java program opens the NetCDF file and using a NetCDF support library, extracts the data defined by the user rectangle.
3. The extracted latitude and longitude data are then converted into the PostGIS geometry format, while the depth data remain unchanged.
4. The Java program then opens a JDBC connection to the PostgreSQL PDB database. Connection parameters are defined in the top section of the file.
5. The Java program then deletes any remaining rows in the ETOPO2 table, owned by that particular PostgreSQL database user.
6. The Java program then uses the PostgreSQL JDBC Driver library to send the extracted data to the database table.

7.3 User-exit installation

All the files, programs and packages related to the user-exit can be found in the `user_exit` directory on the DVD. You must first install PL/PerlU on your PostgreSQL installation. Execute the following command:

```
sudo apt-get install postgresql-plperl-8.3
```

You will have to adjust the version of the package to match the version of your PostgreSQL installation. For those with no Internet connection, we included the PL/PerlU package on the DVD (both for i386 and 64bit computers). Next, make the new language package available for your database. This is done with either the command:

```
createlang plperlu template1
```

or within the PostgreSQL environment, connecting via `psql -U postgres pdb`:

```
CREATE LANGUAGE plperlu;
```

This last SQL command only applies for the selected database and will be lost if the database is dropped. The first command (using the template) will be automatically applied to all databases created from the template.

Before loading the user-exit function, edit the upper part of the file to adjust any directory path and version numbers according to your set up environment. The user-exit PL/PerlU code can be found in Annex Q. The user-exit code can now be loaded to the PDB:

```
psql -U postgres pdb < user_exit_fct.sql
```

Next, you will need to compile the external java code. The Java code for the external program called by the user-exit can be found in Annex R. The Java program requires three open-source Java libraries:

1. The NetCDF supporting library for Java `ncCore-4.1.jar` required for parsing NetCDF files (the minimal core version was sufficient for the purpose of this contract). Downloaded from: `http://www.unidata.ucar.edu/software/netcdf-java/`.
2. The library from the Java Topology Suite (JTS) `jts-1.8.jar` required for creating and manipulating GIS objects. Downloaded from: `http://www.vividsolutions.com/jts/jtshome.htm`
3. The JDBC PostrgreSQL driver `postgresql-8.3-605.jdbc4.jar` required for opening a connection to a PostgreSQL database and sending SQL commands. Downloaded from: `http://jdbc.postgresql.org/download/postgresql-8.3-605.jdbc4.jar` All these libraries are provided in the directory `user_exit` on the DVD; again, for those with no Internet connection.

We recommend to use Java 1.6 from Sun (`/usr/lib/jvm/java-6-sun`). Before compiling, edit the top of the Java file `copyETOPO2DataToPDB.java` to adjust the database settings and the filepath of the NetCDF ETOPO2 file to fit your working environment.

```
javac -cp .:/install_dir/user_exit/postgresql-8.3-605.jdbc4.jar:
/install_dir/user_exit/ncCore-4.1.jar:/install_dir/user_exit/jts-1.8.jar
 copyETOPO2DataToPDB.java
```

You will also need to adjust the above classpath so the libraries are recognized by the compiler. After successful compilation, you will need to verify that:

1. NetCDF ETOPO2 file has been uncompressed : `unzip ETOPO2v2c_f4_netCDF.zip` and that its path is the same as set in `copyETOPO2DataToPDB.java`.
2. The user-exit will run the external program as user `postgres`. Thus, read permission must be set along all the directory path of the file to be read and path of external libraries to be loaded.

Before running the program from PostgreSQL, you can test the program by running it directly, at the command line in your UNIX or Windows environment. The java program has the necessary tool to connect to your PDB database remotely. Note the program needs four arguments as input: the latitude and longitude of the upper left corner of your rectangle and the latitude and longitude of the lower right corner, respectively.

```
java -cp .:/install_dir/user_exit/postgresql-8.3-605.jdbc4.jar:
/install_dir/user_exit/ncCore-4.1.jar:/install_dir/user_exit/jts-1.8.jar
 copyETOPO2DataToPDB 41.0 -150.0 40.0 -145.0
```

Note the "." (current directory) in the library path. After execution, the above command will have created 4500 rows in your ETOPO2 table. Note that on each execution, **the Etopo2 table is first emptied before being filled with the new results**. If time permits, a better implementation would have been to pass the database username (i.e., *postgres*) along with the 4 parameters so the external program could resend the results to a **temporary** Etopo2 table created by that user. This would allow multiple ETOPO2 tables, one for each PDB user. The table deletion would be taken care of by the temporary status (feature provided by PostgreSQL) of the table which are removed at the end of the session, providing an automatic cleaning of the DBMS environment. The current setup does not allow multiple database users, each with their own ETOPO2 table.

Now that you tested the user-exit from the UNIX/Windows command line, you can now test the complete user-exit, within the database PostgreSQL command line. First connect to the PDB database:

```
psql -U postgres pdb
```

Then execute the user-exit function, providing as input the latitude and longitude of the upper left corner of your rectangle and the latitude and longitude of the lower right corner. Be aware that a very large area may take a lot of time to compute as well as potentially resulting in an `Out of memory` situation.

```
SELECT loadEtopo2Table(41.0, -150.0, 40.0, -145.0);
```

When the function returns, you can verify that the Etopo2 table has been filled with the subset data. The input parameters are tested for inconsistencies before passing them to the Java program. The function returns when finished. Performance is reasonable, around 100 sec for a rectangle 10 degrees(latitude) by 20 degrees(longitude) producing 180,000 rows in the ETOPO2 table. Performance could be improved in the external Java code.

Now that the proof-of-concept user-exit has been completed, other user-exits can be built based on this model. Nevertheless, we have to keep in mind that the process is still experimental and must be evaluated on a case by case basis.

T8- Improvement to the bathymetric user-exit (optimization) (8 hours).

T9- Implementation of other user-exits (40 to 160 hours).

8 Lineage data

Unfortunately, time and resources were insufficient to complete the implementation of the lineage data capture. For this contract, priority was given to the successful implementation of LDB to PDB porting and the user-exit mechanism.

The design for this section was to add other tabs to the GUI interface in order to capture and validate the potentially complex structure of the lineage data. The GUI would have provided a clean process for filling up the information and querying the user for any inconsistency or missing mandatory fields. The GUI connection to the PDB for filling up lineage tables would have been identical to the JDBC mechanism used in the user-exit.

The delay in the implementation of the lineage data capture could be viewed as a good opportunity for rethinking the issues that need to be resolved before adding data to these tables.

Q6- The definition of what is a package needs to be clarified. Is the PDB is a single data package or is it made up of many packages?
Q7- Can a data package be composed of many data packages? For example XBT and the many cruises which produced the data.
Q8- How will these definitions impact the lineage tables?
Q9- How to manage (in the GUI or elsewhere) multiple data packages?

The evaluation period could provide new insights for future development.

T10- Completion of capturing lineage data (40 to 80 hours).

9 Conclusions

In this project, we successfully created SQL scripts for porting the legacy LDB database to the new REA production database. Although the original contract did not require porting all the datasets, we were able to do so within the time frame, as well as including the Scotia Shelf Sediment data set mentioned in Isenor and Spears [1]. The scripts are manageable in size and were documented.

The user-exit proof of concept was successfully implemented for the bathymetric data using Java code and a PL/PerlU script. This opens the way for other user-exit implementations (e.g., sea bed forms).

On the other hand, the implementation of the lineage data capture was sacrificed to the profit of porting all the datasets. Because of lack of time and resources, only the first phase of the GUI was implemented. The second phase, still to be done, would have seen the lineage data capture addition to the GUI prototype.

The validation process was only implemented partially. Because the porting process involves a lot of data manipulation and casting from string to numbers, the end results are sometimes quite different from the original inputs, making the comparison difficult. More complex validation functions would be needed for exact and complete comparison; but priorities were placed elsewhere. Another possibility would be to implement partial validation based on some data characteristic that did not change during the porting process.

Some filtering features were added for the XBT/XSV and NADAS dataset to allow updates of the PDB from newer releases of the LDB. Using a timestamp, only newer values of these two datasets can be transfered to the PDB. Tests performed within the LDB were successful but no real test sets were available to fully test this new functionality. It is still the user responsibility to make sure that all the parameters needed for the porting of the XBT/XSV and NADAS datasets are available (such as related cruise data and possibly other related data).

Some smaller but non-trivial issues still remain:
– Porting GUI improvement (proper sequential behaviour)
– Performance issues to understand and resolve in specific areas
– Adding more tests and functions to validate the data manipulations from start to end.
– Test the filtering with other datasets and add modifications to include dependent parameters (e.g., cruise data)

References

[1] Isenor, A.W. and Spears, T.W. (2009), Utilizing Arc Marine concepts for designing a geospatially enabled database to support rapid environmental assessment, (DRDC Atlantic TM 2009-061) Defence R&D Canada – Atlantic.

[2] Deveau, TJ (2008), Rapid Environmental Assessment Database Project-Phase I, (DRDC Atlantic CR 2008-044) Defence R&D Canada – Atlantic.

This page intentionally left blank.

Annex A: sequences.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- The following sequences were defined to be associated with the default value (see alterations.sql)
-- of the ID for feature, login, ship, scientist, measurement_location, area_characteristic, station,
-- survey, series and etopo2 tables. These sequences are rather elementary but are more than
-- sufficient for the porting of the LDB to the PDB. In the case of updating the PDB with
-- additional data, one should revisit the sequence scheme. It is possible that the following
-- sequences could still be valid but it would depend of the update elements.

-- feature_id sequence
CREATE SEQUENCE feature_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE feature_id_seq OWNER TO postgres;

-- login_id sequence
CREATE SEQUENCE login_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE login_id_seq OWNER TO postgres;

-- ship_id sequence
CREATE SEQUENCE ship_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE ship_id_seq OWNER TO postgres;

-- scientist_id sequence
CREATE SEQUENCE scientist_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE scientist_id_seq OWNER TO postgres;

-- measurement_location_id sequence
CREATE SEQUENCE measurement_location_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
```

```
   CACHE 1;
ALTER TABLE measurement_location_id_seq OWNER TO postgres;

-- area_characteristic_id sequence
CREATE SEQUENCE area_characteristic_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE area_characteristic_id_seq OWNER TO postgres;

-- station_id sequence
CREATE SEQUENCE station_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE station_id_seq OWNER TO postgres;

-- survey_id sequence
CREATE SEQUENCE survey_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE survey_id_seq OWNER TO postgres;

-- series_id sequence
CREATE SEQUENCE series_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE series_id_seq OWNER TO postgres;

-- etopo2_id sequence
CREATE SEQUENCE etopo2_id_seq
   INCREMENT 1
   START 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   CACHE 1;
ALTER TABLE etopo2_id_seq OWNER TO postgres;
```

Annex B: alterations.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- These are the ALTER TABLE commands applied to the data model necessary for the porting
-- of the data from the LDB to the PDB. Most of them are assignments of a DEFAULT value
-- (ex. SEQUENCE function to an ID). Some of them drop some predefined constraints imposed on
-- geometry object and should be investigated further to see if these constraint still
-- applies as the data model has evolved since the beginning.
-- For example, the ALTER TABLE for the removal of the SRID constraint may not be necessary
-- anymore as we recently uniformaly set the SRID to 4326 throughout the PDB.
-- The TRIGGER functions for removing automatically the source feature_id when
-- deleting the feature_id of instantaneous_point or mesh_point has been deactivated (commented)
-- because of very poor performence issues in large tables.
-- A better implementation of counter default value should be investigated. Setting the default
-- value to "1" was sufficient for the initial porting but may not be adequate when adding update
-- to the PDB. In this latter case, counters should be set to the latest value (from the PDB) plus one
-- before proceeding to the update.


-- feature
-- Assignment of a DEFAULT value for feature_id from the next value of a the corresponding sequence
ALTER TABLE feature ALTER COLUMN feature_id SET DEFAULT nextval('feature_id_seq'::regclass);

-- login table
-- Assignment of a DEFAULT value for login_id from the next value of a the corresponding sequence
ALTER TABLE login ALTER COLUMN login_id SET DEFAULT nextval('login_id_seq'::regclass);

-- ship table
-- Assignment of a DEFAULT value for ship_id from the next value of a the corresponding sequence
ALTER TABLE ship ALTER COLUMN ship_id SET DEFAULT nextval('ship_id_seq'::regclass);

-- cruise_note table
-- Setting the default value of a new note_counter to one
ALTER TABLE cruise_note ALTER COLUMN note_counter SET DEFAULT '1';

-- scientist table
-- Assignment of a DEFAULT value for scientist_id from the next value of a the corresponding sequence
ALTER TABLE scientist ALTER COLUMN scientist_id SET DEFAULT nextval('scientist_id_seq'::regclass);

-- scientist_on_cruise table
-- Setting the default value of scientist_counter to one
ALTER TABLE scientist_on_cruise ALTER COLUMN scientist_counter SET DEFAULT '1';

-- ships_on_cruise table
-- Setting the default value of ship_counter to one
ALTER TABLE ships_on_cruise ALTER COLUMN ship_counter SET DEFAULT '1';

-- measurement_location table
-- Assignment of a DEFAULT value for measurement_location_id from the next value of a the corresponding sequence
ALTER TABLE measurement_location ALTER COLUMN measurement_location_id SET DEFAULT
  nextval('measurement_location_id_seq'::regclass);
-- Removing the constraint that forces the DIMS to be the same as the default (2)
```

```
ALTER TABLE measurement_location DROP CONSTRAINT "enforce_dims_geom";
-- Removing the constraint that forces the SRID to be the same as the default (4326)
ALTER TABLE measurement_location DROP CONSTRAINT "enforce_srid_geom";

-- area_characteristic table
-- Assignment of a DEFAULT value for area_characteristic_id from the next value of a the corresponding sequence
ALTER TABLE area_characteristic ALTER COLUMN area_characteristic_id SET DEFAULT
  nextval('area_characteristic_id_seq'::regclass);

-- feature_area table
-- Removing the constraint that forces the geotype to be the same as the default (POINT)
ALTER TABLE feature_area DROP CONSTRAINT "enforce_geotype_geom";
-- Removing the constraint that forces the SRID to be the same as the default (4326)
ALTER TABLE feature_area DROP CONSTRAINT "enforce_srid_geom";
-- The following trigger function was design to automatically delete the feature associated
-- to a feature_area but it was not implemented because of very important
-- performance issues.
-- CREATE TRIGGER triggerFeatureDelete AFTER DELETE ON feature_area
-- FOR EACH ROW EXECUTE PROCEDURE triggerFeatureDelete();

-- mesh_point table
-- Removing the constraint that forces the SRID to be the same as the default (4326)
ALTER TABLE mesh_point DROP CONSTRAINT "enforce_srid_geom";
-- Removing the constraint that forces the DIMS to be the same as the default (2)
ALTER TABLE mesh_point DROP CONSTRAINT "enforce_dims_geom";
-- The following trigger function was design to automatically delete the feature associated
-- to a mesh_point but it was not implemented because of very important
-- performance issues.
-- CREATE TRIGGER triggerFeatureDelete AFTER DELETE ON mesh_point
-- FOR EACH ROW EXECUTE PROCEDURE triggerFeatureDelete();
-- DROP TRIGGER triggerFeatureDelete ON table_name;

-- survey table
-- Assignment of a DEFAULT value for survey_id from the next value of a the corresponding sequence
ALTER TABLE survey ALTER COLUMN survey_id SET DEFAULT nextval('survey_id_seq'::regclass);

-- instantaneous_point table
-- The following trigger function was design to automatically delete the feature associated
-- to an instantaneous_point but it was not implemented because of very important
-- performance issues.
-- CREATE TRIGGER triggerFeatureDelete AFTER DELETE ON instantaneous_point
-- FOR EACH ROW EXECUTE PROCEDURE triggerFeatureDelete();

-- series table
-- Assignment of a DEFAULT value for series_id from the next value of a the corresponding sequence
ALTER TABLE series ALTER COLUMN series_id SET DEFAULT nextval('series_id_seq'::regclass);

-- etopo2 table
-- Removing the constraint that forces the SRID to be the same as the default (4326)
ALTER TABLE etopo2 DROP CONSTRAINT "enforce_srid_geom";
-- Assignment of a DEFAULT value for etopo2_id from the next value of a the corresponding sequence
ALTER TABLE etopo2 ALTER COLUMN etopo2_id SET DEFAULT nextval('etopo2_id_seq'::regclass);
```

Annex C: util_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- createFeature
-- Function which creates a new row in the feature table associate to a given data_package_id
-- and returns the corresponding feature_id
CREATE OR REPLACE FUNCTION createFeature(input_name char(20), input_package_id integer)
RETURNS integer AS
$BODY$
 DECLARE
   f_id integer;
 BEGIN
  INSERT INTO pdb.public.feature(feature_class, data_package_id)
    VALUES (input_name, input_package_id) RETURNING feature_id  INTO f_id;
  RETURN f_id;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION createFeature(char(20), integer) OWNER TO postgres;

-- getscientistname
-- Function that retrieves the scientist name in the readbv3b DB from a given ID number
CREATE OR REPLACE FUNCTION getScientistNameFromID(scientistid integer)
RETURNS  char(100) AS
$BODY$
  DECLARE
    rec RECORD;
  BEGIN
    SELECT * INTO rec FROM dblink('dbname=readbv3b user=postgres','SELECT scientist FROM
      readbv3b.public.scientists WHERE id = '||scientistid) AS db(db_id char(100));
  RETURN rec.db_id;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getScientistNameFromID(integer) OWNER TO postgres;

-- getscientistid
-- Function that retrieves the scientist ID in the PDB from a given scientist name
CREATE OR REPLACE FUNCTION getScientistIDFromName(tmp_name char(100))
RETURNS integer AS
$BODY$
  DECLARE
    rec RECORD;
  BEGIN
    SELECT scientist_id INTO rec FROM pdb.public.scientist WHERE scientist_name ~ tmp_name ;
    RETURN rec.scientist_id;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getScientistIDFromName(char(100)) OWNER TO postgres;

-- getCruiseIDFromIDFilename
```

```
-- Function that returns the cruise number from the readbv3b filenames ID
CREATE OR REPLACE FUNCTION getCruiseIDFromIDFilename(filenameid integer)
RETURNS  integer AS
$BODY$
  DECLARE
    rec RECORD;
  BEGIN
    SELECT * INTO rec FROM dblink('dbname=readbv3b user=postgres','SELECT id_cruise FROM
      readbv3b.public.filenames WHERE id = '||filenameid) AS db(db_id INTEGER);
    RETURN rec.db_id;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getCruiseIDFromIDFilename(integer) OWNER TO postgres;

-- getmeasuringdeviceid
-- Function that retrieves the measuring device ID in the PDB from a given device name
-- This function was necessary because of some discrepancies between the probe name from the LDB
-- and the device name chose in the PDB
CREATE OR REPLACE FUNCTION getMeasuringDeviceIDFromName(tmp_name char(100))
RETURNS integer AS
$BODY$
  DECLARE
    rec RECORD;
  BEGIN
--  In general, the probe_type is sufficient for detecting the same device in the PDB device list
    SELECT measuring_device_id INTO rec FROM pdb.public.measuring_device WHERE description ~ tmp_name ;
-- Except for a few exceptions:
    IF tmp_name = 'SEA-BIRD SBE19PLUS DATA FILE:' THEN
--    Device to be selected is SBE19PLUS
      SELECT measuring_device_id INTO rec FROM pdb.public.measuring_device WHERE description ~ 'SBE19PLUS' ;
    END IF;
--    Device to be selected is XBTT7
    IF tmp_name = 'T-7' THEN
      SELECT measuring_device_id INTO rec FROM pdb.public.measuring_device WHERE description ~ 'T-7' AND name <> 'XBTT7DB' ;
    END IF;
--    Device to be selected is XBTT7DB
    IF tmp_name ~ 'DB' THEN
      SELECT measuring_device_id INTO rec FROM pdb.public.measuring_device WHERE description ~ 'T-7' AND name = 'XBTT7DB' ;
    END IF;
    RETURN rec.measuring_device_id;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getMeasuringDeviceIDFromName(char(100)) OWNER TO postgres;

-- createTimestamp
-- Function that create and returns an SQL timestamp from an input date and time
CREATE OR REPLACE FUNCTION createTimestamp(tmp_date text, tmp_time text)
RETURNS  timestamp without time zone AS
$BODY$
  DECLARE
    tmp_str char(20);
  BEGIN
    tmp_str = tmp_date||tmp_time;
    RETURN to_timestamp(tmp_str,'YYMMDDHHMISS');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION createTimestamp(text,text) OWNER TO postgres;

-- getLatFromPointRangeBearing
-- Function that returns the coordinates (latitude,longitude) of a point located
-- at a range and a bearing from a given starting point (start_lat,start_long)
```

```
-- This function is use to calculate the coordinate of the apexes of equilateral triangle
-- from its center in the insertECS SQL function.
CREATE OR REPLACE FUNCTION getCoordFromPointRangeBearing(start_lat double precision,
  start_long double precision, bearing double precision, range double precision)
RETURNS  decimal[] AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_bearing decimal;
    tmp_angular_dist decimal;
    tmp_startLatRad decimal;
    tmp_startLngRad decimal;
    tmp_dlon decimal;
    tmp_long decimal;
    tmp_lat decimal;
  BEGIN
--  Converting degrees into radians (minus is added for cancelling the minus of the West longitude)
--  Everything could be still be good without the minus (to be verified)
--  See http://www.movable-type.co.uk/scripts/latlong.html for a complete description of the formula
    tmp_startLatRad = (pi()/180.0) * start_lat;
    tmp_startLngRad = -1.0*(pi()/180.0) * start_long;
    tmp_bearing = bearing * (pi()/180.0);
--  Angular distance in radian using 360*60 nautical miles as for the radius of the earth
    tmp_angular_dist = range * (pi()/ (180.0*60.0));

--  Computation of the new latitude
    tmp_lat = asin(sin(tmp_startLatRad)*cos(tmp_angular_dist)+cos(tmp_startLatRad)*sin(tmp_angular_dist)*cos(tmp_bearing));
--  Computation of the delta in the longitude
    tmp_dlon = atan2(sin(tmp_bearing)*sin(tmp_angular_dist)*cos(tmp_startLatRad),cos(tmp_angular_dist)
      -sin(tmp_startLatRad)*sin(tmp_lat));
--  Computation of the longitude and adjusting for to fit the usual longitude interval
    tmp_long =(cast(tmp_startLngRad - tmp_dlon + pi() AS decimal))%(cast((2.0*pi()) AS decimal)) - pi();

--  Converting back the radian into degrees
    tmp_lat = (180.0/pi()) * tmp_lat;
    tmp_long = (180.0/pi()) * tmp_long;
--  Adjusting for the longitude sign and keeping the result higher than -180.0
    IF ( tmp_long < -180.0 ) THEN
      tmp_long = (360.0 + tmp_long );
    END IF;

    RETURN ARRAY[tmp_lat, (-1)*tmp_long];
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getCoordFromPointRangeBearing(double precision, double precision,
  double precision, double precision) OWNER TO postgres;

-- triggerFeatureDelete()
-- Trigger function that deletes the feature_id in the feature table when deleting
-- the corresponding feature_id in table instantaneous_point or mesh_point.
-- This function was not used because of big performance issues in large tables.
CREATE OR REPLACE FUNCTION triggerFeatureDelete() RETURNS TRIGGER AS $triggerFeatureDelete$
BEGIN
  DELETE FROM feature WHERE feature_id = OLD.feature_id;
  RETURN NULL;
END;
$triggerFeatureDelete$ LANGUAGE plpgsql;
ALTER FUNCTION triggerFeatureDelete() OWNER TO postgres;


-- getTime
-- Function that returns a string containing the pourcentage of the row already processed
-- and time left before terminating the insertAAA command to terminate. It uses as inputs,
```

```
-- the current row number being processed, the total number of rows of the table being processed,
-- and the initial timestamp when the processing of that table started.
CREATE OR REPLACE FUNCTION printTimeLeft(current_row_nb integer, total_row_nb integer,
  start_time timestamp without time zone)
RETURNS varchar(200) AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_time_spent double precision;
    tmp_time_left double precision;
    tmp_str varchar(200);
  BEGIN
--  calculating the time spent between now and when execution of the insertAAA started
    tmp_time_spent = EXTRACT(EPOCH FROM (clock_timestamp()-start_time));
--  calculating (approximately) the time left before the termination of the insertAAA function
    tmp_time_left = tmp_time_spent*total_row_nb/current_row_nb - tmp_time_spent;
--  preparing the string with the row number info, % done, and the time left
    tmp_str = 'At row '||current_row_nb||' of '||total_row_nb||', '||
        (100 - (((total_row_nb - current_row_nb)*100)/total_row_nb))||'% done, '||
        trunc(tmp_time_left/3600)||' hr and '|| trunc((CAST(tmp_time_left AS integer)%3600)/60 )||' min left.';
    RETURN tmp_str;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION printTimeLeft(integer,integer, timestamp without time zone) OWNER TO postgres;
```

Annex D: login_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- Function that reads the username and password columns of the LDB logins table and copy
-- the content to the same columns in the PDB.
CREATE OR REPLACE FUNCTION insertLogin()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Login: Inserting login...';
    INSERT INTO pdb.public.login(password,username) SELECT * FROM
      dblink('dbname=readbv3b user=postgres','SELECT password,username FROM readbv3b.public.logins')
      AS t(a text, b text);
  END;
$BODY$
   LANGUAGE 'plpgsql';
ALTER FUNCTION insertLogin() OWNER TO postgres;

-- Function that compare the username and password columns of the LDB logins table
-- with the login table in the PDB.
CREATE OR REPLACE FUNCTION verifyLogin()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
--  In this block, we verify that all the usernames in the LDB appear also
--  in the PDB username column
    RAISE NOTICE 'Verifying Login...';
    SELECT count(*) INTO tmp_rec FROM (SELECT username FROM login EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT username FROM readbv3b.public.logins')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (login)';
    ELSE
       RAISE NOTICE 'Missmatch found in login(username)';
    END IF;

--  In this block, we verify that all the passwords in the LDB appear also
--  in the PDB password column
    SELECT count(*) INTO tmp_rec FROM (SELECT password FROM login EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT password FROM readbv3b.public.logins')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (password)';
    ELSE
       RAISE NOTICE 'Missmatch found in login(password)';
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
```

```
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyLogin() OWNER TO postgres;

-- Function that remove all rows from the login table.
CREATE OR REPLACE FUNCTION deleteLogin()
RETURNS VOID AS
 $BODY$
  BEGIN
    RAISE NOTICE 'Deleting login...';
    DELETE FROM login;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteLogin() OWNER TO postgres;
```

Annex E: instrument_mngt_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- Function that creates the measuring_device table and device_class_detail table in the PDB
-- Most of the values were based from the PDB design report.
CREATE OR REPLACE FUNCTION insertInstrumentManagement()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Instrument Mngt: Inserting measuring device id...';
    INSERT INTO pdb.public.measuring_device(measuring_device_id, name, description) VALUES
      (1,'COMPUTED', 'Value originated from a simulation or an algorithm'),
      (2, 'XBTT5', 'Expendable Bathythermograph model T-5'),
      (3, 'XBTT7', 'Expendable Bathythermograph model T-7'),
      (4, 'XSV02', 'Expendable Sound Velocimeter model XSV-02'),
      (5, 'SBE19PLUS','SEACAT SBE19PLUS Profiler, Conductivity, Temperature, and Pressure Recorder'),
      (6, 'SONOBUOY', 'Sonobuoy'),
      (7, 'NADAS', 'Device id associated to all NADAS measurement'),
      (8, 'XBTT7DB', 'Expendable Bathythermograph model T-7(DB)');

--  The Terminal depth values come from the xbt_file_meta_data table (terminal_depth_m column)
    INSERT INTO pdb.public.device_class_detail(measuring_device_id, descriptor, value) VALUES
      (2, 'XBTTerminalDepth',1830),
      (3, 'XBTTerminalDepth', 760),
      (4, 'XBTTerminalDepth', 2000),
      (5, 'XBTTerminalDepth', 0),
      (8, 'XBTTerminalDepth', 760);
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertInstrumentManagement() OWNER TO postgres;

-- Function that verifies that all the measuring_device and device_class_detail are in place in the PDB
CREATE OR REPLACE FUNCTION verifyInstrumentManagement()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Instrument Management ...';
--  Verifying measuring_device_id = 1 : COMPUTED
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 1;
    IF tmp_rec.name =  'COMPUTED' THEN
       RAISE NOTICE 'OK (measuring_device: COMPUTED)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:COMPUTED';
    END IF;
--  Verifying measuring_device_id = 2 : XBTT5
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 2;
    IF tmp_rec.name =  'XBTT5' THEN
       RAISE NOTICE 'OK (measuring_device: XBTT5)';
```

```
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:XBTT5';
    END IF;
--  Verifying measuring_device_id = 3 : XBTT7
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 3;
    IF tmp_rec.name =  'XBTT7' THEN
       RAISE NOTICE 'OK (measuring_device: XBTT7)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:XBTT7';
    END IF;
--  Verifying measuring_device_id = 4 : XSV02
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 4;
    IF tmp_rec.name =  'XSV02' THEN
       RAISE NOTICE 'OK (measuring_device: XSV02)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:XSV02';
    END IF;
--  Verifying measuring_device_id = 5 : SBE19PLUS
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 5;
    IF tmp_rec.name =  'SBE19PLUS' THEN
       RAISE NOTICE 'OK (measuring_device: SBE19PLUS)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:SBE19PLUS';
    END IF;
--  Verifying measuring_device_id = 6 : SONOBUOY
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 6;
    IF tmp_rec.name =  'SONOBUOY' THEN
       RAISE NOTICE 'OK (measuring_device: SONOBUOY)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:SONOBUOY';
    END IF;
--  Verifying measuring_device_id = 7 : NADAS
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 7;
    IF tmp_rec.name =  'NADAS' THEN
       RAISE NOTICE 'OK (measuring_device: NADAS)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:NADAS';
    END IF;
--  Verifying measuring_device_id = 8 : XBTT7DB
    SELECT name INTO tmp_rec FROM measuring_device WHERE measuring_device_id = 8;
    IF tmp_rec.name =  'XBTT7DB' THEN
       RAISE NOTICE 'OK (measuring_device: XBTT7DB)';
    ELSE
       RAISE NOTICE 'Missmatch found in measuring_device:XBTT7DB';
    END IF;
--  Verifying device_class_detail, measuring_device_id = 2 : XBTT5
    SELECT descriptor,value INTO tmp_rec FROM device_class_detail WHERE measuring_device_id = 2;
    IF tmp_rec.descriptor =  'XBTTerminalDepth' AND tmp_rec.value = 1830 THEN
       RAISE NOTICE 'OK (device_class_detail: XBTT5)';
    ELSE
       RAISE NOTICE 'Missmatch found in device_class_detail: measuring_device_id = 2';
    END IF;
--  Verifying device_class_detail, measuring_device_id = 3 : XBTT7
    SELECT descriptor,value INTO tmp_rec FROM device_class_detail WHERE measuring_device_id = 3;
    IF tmp_rec.descriptor =  'XBTTerminalDepth' AND tmp_rec.value = 760 THEN
       RAISE NOTICE 'OK (device_class_detail: XBTT7)';
    ELSE
       RAISE NOTICE 'Missmatch found in device_class_detail: measuring_device_id = 3';
    END IF;
--  Verifying device_class_detail, measuring_device_id = 4 : XSV02
    SELECT descriptor,value INTO tmp_rec FROM device_class_detail WHERE measuring_device_id = 4;
    IF tmp_rec.descriptor =  'XBTTerminalDepth' AND tmp_rec.value = 2000 THEN
       RAISE NOTICE 'OK (device_class_detail: XSV02)';
    ELSE
```

```
       RAISE NOTICE 'Missmatch found in device_class_detail: measuring_device_id = 4';
    END IF;
--  Verifying device_class_detail, measuring_device_id = 5 : SBE19PLUS
    SELECT descriptor,value INTO tmp_rec FROM device_class_detail WHERE measuring_device_id = 5;
    IF tmp_rec.descriptor =  'XBTTerminalDepth' AND tmp_rec.value = 0 THEN
       RAISE NOTICE 'OK (device_class_detail: SBE19PLUS)';
    ELSE
       RAISE NOTICE 'Missmatch found in device_class_detail: measuring_device_id = 5';
    END IF;
--  Verifying device_class_detail, measuring_device_id = 8 : XBTT7DB
    SELECT descriptor,value INTO tmp_rec FROM device_class_detail WHERE measuring_device_id = 8;
    IF tmp_rec.descriptor =  'XBTTerminalDepth' AND tmp_rec.value = 760 THEN
       RAISE NOTICE 'OK (device_class_detail: XBTT7DB)';
    ELSE
       RAISE NOTICE 'Missmatch found in device_class_detail: measuring_device_id = 8';
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyInstrumentManagement() OWNER TO postgres;


-- Function which deletes the measuring_device table and the device_class_detail table.
CREATE OR REPLACE FUNCTION deleteInstrumentManagement()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting device_class_detail...';
    DELETE FROM device_class_detail;
    RAISE NOTICE 'Deleting measuring_device...';
    DELETE FROM measuring_device;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteInstrumentManagement() OWNER TO postgres;
```

This page intentionally left blank.

Annex F: parameter_mgnt_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- Function that creates the parameter table and quality_flag table in the PDB
-- The values were based from the PDB design report (value 50 to 322, and 617).
-- Other values (400 and 800 series), were custum-made defined ID for the purpose
-- of completing the design.
CREATE OR REPLACE FUNCTION insertParameterManagement()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Parameter Mngt: Inserting parameter...';
    INSERT INTO pdb.public.parameter(parameter_id, name, description, quantity, units) VALUES
(50, 'Surface Water Temperature', 'Surface Water Temperature', 2, 'C'),
(51, 'Relative Humidity', 'Relative Humidity', 2, '%' ),
(52, 'Barometric Pressure', 'Barometric Pressure', 2, 'mbar'),
(53, 'Precipitation', 'Precipitation', 2, 'mm'),
(55, 'Air Temperature', 'Air Temperature', 2, 'C'),
(57, 'Solar Visibility', 'Solar Visibility', 2, 'Watt/m2'),
(127, 'Ship Heading', 'Ship Heading', 2, 'deg'),
(128, 'Wind Speed', 'Wind Speed', 2, 'knots'),
(129, 'Wind Direction', 'Wind Direction', 2, 'deg'),
(130, 'Rotation frequency of port side propeller', 'Rotation frequency of port side propeller', 2, 'rpm'),
(131, 'Rotation frequency of starboard side propeller', 'Rotation frequency of starboard side propeller',
             2, 'rpm'),
(134, 'Ship Speed Over Ground', 'Ship Speed Over Ground', 2, 'knots'),
(135, 'Ship Course Over Ground','Ship Course Over Ground', 2, 'deg'),
(308, 'Water Depth', 'Water Depth', 2, 'm'),
(322, 'Water Temperature','Water Temperature', 2, 'C'),
(401, 'Sediment Thickness','Sediment Thickness', 2, 'm'),
(402, 'Salinity','Salinity', 2, 'ppm'),
(617, 'Sound Speed', 'Sound Speed', 2, 'm/s'),
(800, 'Wind Estimate', 'Wind Estimate', 2, 'knots'),
(801, 'Ambient Noise Mean', 'Ambient Noise Mean', 2, 'dB re 1 micro Pascal^2/Hz'),
(802, 'Ambient Noise Standard Deviation', 'Ambient Noise Standard Deviation', 2, 'dB re 1 micro Pascal^2/Hz'),
(803, 'Group Number', 'Group Number, identical to id (sediment_ss_position)', 2, ''),
(804, 'Number of measurements', 'Number of measurements', 2, ''),
(805, 'Mean Velocity of sound in the bottom', 'Mean Velocity of sound in the bottom', 2, 'km/s'),
(806, 'Error associated with the mean velocity', 'Error associated with the mean velocity', 2, 'km/s'),
(807, 'High Cluster', 'The number of points in a cluster which exists more than one standard deviation'||
             ' from the mean, on the side of increased velocity', 2, ''),
(808, 'Low Cluster', 'The number of points in a cluster which exists more than one standard deviation'||
             ' from the mean, on the side of decreased velocity', 2, ''),
(809, 'Average Clearance', 'average distance of the measuring equipment from the seabed', 2, 'm'),
(810, 'Central Frequency', 'Ambient Noise Central Frequency', 2, 'Hz'),
(811, 'Range', 'Range', 2, 'Km'),
(812, 'Shot Number', 'Shot Number', 2, ''),
(813, 'Run direction', 'Run direction', 2, 'deg'),
(814, 'Tranloss', 'Transmission Loss', 2, 'dB'),
(815, 'Central Frequency', 'Transmission Loss Central Frequency', 2, 'Hz');

--  Only one quality flag was used in this contract: "unknown". Other quality flag could be added to this
```

```
--  table in the future.
    INSERT INTO pdb.public.quality_flag(quality_flag_id, short_description, long_description, secondary_flag,
      source, source_flag) VALUES (0,'unknown','unknown','unknown','unknown','unknown');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertParameterManagement() OWNER TO postgres;

-- Function that verifies that all the parameters and quality_flag are in place in the PDB
CREATE OR REPLACE FUNCTION verifyParameterManagement()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Parameter Management...';
--  Verification of parameter number 50: Surface Water Temperature
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 50;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Surface Water Temperature' THEN
       RAISE NOTICE 'OK (parameter: Surface Water Temperature)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Surface Water Temperature';
    END IF;
--  Verification of parameter number 51: Relative Humidity
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 51;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Relative Humidity' THEN
       RAISE NOTICE 'OK (parameter: Relative Humidity)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Relative Humidity';
    END IF;
--  Verification of parameter number 52: Barometric Pressure
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 52;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Barometric Pressure' THEN
       RAISE NOTICE 'OK (parameter: Barometric Pressure)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Barometric Pressure';
    END IF;
--  Verification of parameter number 53: Precipitation
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 53;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Precipitation' THEN
       RAISE NOTICE 'OK (parameter: Precipitation)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Precipitation';
    END IF;
--  Verification of parameter number 55: Air Temperature
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 55;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Air Temperature' THEN
       RAISE NOTICE 'OK (parameter: Air Temperature)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Air Temperature';
    END IF;
--  Verification of parameter number 57: Solar Visibility
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 57;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Solar Visibility' THEN
       RAISE NOTICE 'OK (parameter: Solar Visibility)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Solar Visibility';
    END IF;
--  Verification of parameter number 127: Ship Heading
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 127;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Ship Heading' THEN
       RAISE NOTICE 'OK (parameter: Ship Heading)';
    ELSE
```

```
      RAISE NOTICE 'Missmatch found in parameter: Ship Heading';
   END IF;
--  Verification of parameter number 128: Wind Speed
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 128;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Wind Speed' THEN
      RAISE NOTICE 'OK (parameter: Wind Speed)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Wind Speed';
   END IF;
--  Verification of parameter number 129: Wind Direction
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 129;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Wind Direction' THEN
      RAISE NOTICE 'OK (parameter: Wind Direction)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Wind Direction';
   END IF;
--  Verification of parameter number 130: Rotation frequency of port side propeller
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 130;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Rotation frequency of port side propeller' THEN
      RAISE NOTICE 'OK (parameter: Rotation frequency of port side propeller)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Rotation frequency of port side propeller';
   END IF;
--  Verification of parameter number 131: Rotation frequency of starboard side propeller
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 131;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Rotation frequency of starboard side propeller' THEN
      RAISE NOTICE 'OK (parameter: Rotation frequency of starboard side propeller)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Rotation frequency of starboard side propeller';
   END IF;
--  Verification of parameter number 134: Ship Speed Over Ground
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 134;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Ship Speed Over Ground' THEN
      RAISE NOTICE 'OK (parameter: Ship Speed Over Ground)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Ship Speed Over Ground';
   END IF;
--  Verification of parameter number 135: Ship Course Over Ground
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 135;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Ship Course Over Ground' THEN
      RAISE NOTICE 'OK (parameter: Ship Course Over Ground)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Ship Course Over Ground';
   END IF;
--  Verification of parameter number 308: Water Depth
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 308;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Water Depth' THEN
      RAISE NOTICE 'OK (parameter: Water Depth)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Water Depth';
   END IF;
--  Verification of parameter number 322: Water Temperature
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 322;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Water Temperature' THEN
      RAISE NOTICE 'OK (parameter: Water Temperature)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Water Temperature';
   END IF;
--  Verification of parameter number 401: Sediment Thickness
   SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 401;
   IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Sediment Thickness' THEN
      RAISE NOTICE 'OK (parameter: Sediment Thickness)';
   ELSE
      RAISE NOTICE 'Missmatch found in parameter: Sediment Thickness';
```

```
    END IF;
--  Verification of parameter number 402: Salinity
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 402;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Salinity' THEN
       RAISE NOTICE 'OK (parameter: Salinity)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Salinity';
    END IF;
--  Verification of parameter number 617: Sound Speed
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 617;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Sound Speed' THEN
       RAISE NOTICE 'OK (parameter: Sound Speed)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Sound Speed';
    END IF;
--  Verification of parameter number 800: Wind Estimate
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 800;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Wind Estimate' THEN
       RAISE NOTICE 'OK (parameter: Wind Estimate)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Wind Estimate';
    END IF;
--  Verification of parameter number 801: Ambient Noise Mean
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 801;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Ambient Noise Mean' THEN
       RAISE NOTICE 'OK (parameter: Ambient Noise Mean)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Ambient Noise Mean';
    END IF;
--  Verification of parameter number 802: Ambient Noise Standard Deviation
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 802;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Ambient Noise Standard Deviation' THEN
       RAISE NOTICE 'OK (parameter: Ambient Noise Standard Deviation)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Ambient Noise Standard Deviation';
    END IF;
--  Verification of parameter number 803: Group Number
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 803;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Group Number' THEN
       RAISE NOTICE 'OK (parameter: Group Number)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Group Number';
    END IF;
--  Verification of parameter number 804: Number of measurements
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 804;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Number of measurements' THEN
       RAISE NOTICE 'OK (parameter: Number of measurements)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Number of measurements';
    END IF;
--  Verification of parameter number 805: Mean Velocity of sound in the bottom
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 805;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Mean Velocity of sound in the bottom' THEN
       RAISE NOTICE 'OK (parameter: Mean Velocity of sound in the bottom)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Mean Velocity of sound in the bottom';
    END IF;
--  Verification of parameter number 806: Error associated with the mean velocity
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 806;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Error associated with the mean velocity' THEN
       RAISE NOTICE 'OK (parameter: Error associated with the mean velocity)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Error associated with the mean velocity';
    END IF;
```

```
--  Verification of parameter number 807: High Cluster
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 807;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'High Cluster' THEN
       RAISE NOTICE 'OK (parameter: High Cluster)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: High Cluster';
    END IF;
--  Verification of parameter number 808: Low Cluster
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 808;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Low Cluster' THEN
       RAISE NOTICE 'OK (parameter: Low Cluster)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Low Cluster';
    END IF;
--  Verification of parameter number 809: Average Clearance
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 809;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Average Clearance' THEN
       RAISE NOTICE 'OK (parameter: Average Clearance)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Average Clearance';
    END IF;
--  Verification of parameter number 810: Central Frequency
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 810;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Central Frequency' THEN
       RAISE NOTICE 'OK (parameter: Central Frequency (Ambient Noise))';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Central Frequency (Ambient Noise)';
    END IF;
--  Verification of parameter number 811: Range
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 811;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Range' THEN
       RAISE NOTICE 'OK (parameter: Range)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Range';
    END IF;
--  Verification of parameter number 812: Shot Number
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 812;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Shot Number' THEN
       RAISE NOTICE 'OK (parameter: Shot Number)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Shot Number';
    END IF;
--  Verification of parameter number 813: Run direction
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 813;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Run direction' THEN
       RAISE NOTICE 'OK (parameter: Run direction)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Run direction';
    END IF;
--  Verification of parameter number 814: Tranloss
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 814;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Tranloss' THEN
       RAISE NOTICE 'OK (parameter: Tranloss)';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Tranloss';
    END IF;
--  Verification of parameter number 815: Central Frequency
    SELECT name INTO tmp_rec FROM parameter WHERE parameter_id = 815;
    IF tmp_rec IS NOT NULL AND tmp_rec.name =  'Central Frequency' THEN
       RAISE NOTICE 'OK (parameter: Central Frequency(Tranloss))';
    ELSE
       RAISE NOTICE 'Missmatch found in parameter: Central Frequency(Tranloss)';
    END IF;
    RAISE NOTICE 'Done.';
```

```
--  Verification of quality_flag: unknown
    SELECT short_description, long_description, secondary_flag, source, source_flag
       INTO tmp_rec FROM quality_flag WHERE quality_flag_id = 0;
    IF tmp_rec IS NOT NULL AND tmp_rec.short_description =  'unknown'
                           AND tmp_rec.long_description =  'unknown'
                           AND tmp_rec.secondary_flag =  'unknown'
                           AND tmp_rec.source =  'unknown'
                           AND tmp_rec.source_flag =  'unknown' THEN
       RAISE NOTICE 'OK (quality_flag: unknown )';
    ELSE
       RAISE NOTICE 'Missmatch found in quality_flag: unknown';
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyParameterManagement() OWNER TO postgres;

-- Function which deletes the parameter table and the quality_flag table.
CREATE OR REPLACE FUNCTION deleteParameterManagement()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting parameter...';
    DELETE FROM parameter;
    RAISE NOTICE 'Deleting quality_flag...';
    DELETE FROM quality_flag;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteParameterManagement() OWNER TO postgres;
```

Annex G: position_code_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- Function that creates the position_code table in the PDB
-- The values were based from the design report and other discussions
CREATE OR REPLACE FUNCTION insertPositionCode()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Position code: Inserting position code...';
    INSERT INTO pdb.public.position_code(x_y_position_code, description) VALUES
      ('unknown','unknown'),
      ('computed', 'The XY position was computed from range and bearing'),
      ('Dead Reckoning','Dead Reckoning Position'),
      ('LORAN','LORAN C Fix'),
      ('SAT','Transit Satellite Fix'),
      ('GPS','GPS position'),
      ('Waypoint','Waypoint Position'),
      ('Interpolated','Interpolated using two position fixes on either side of the resulting position values');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertPositionCode() OWNER TO postgres;

-- Function that verifies that all the position codes are in place in the PDB
CREATE OR REPLACE FUNCTION verifyPositionCode()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Position Code ...';
--  Verification of the x_y_position_code: unknown
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'unknown';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'unknown' THEN
      RAISE NOTICE 'OK (Position_Code: unknown)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:unknown';
    END IF;

--  Verification of the x_y_position_code: computed
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'computed';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'The XY position was computed from range and bearing' THEN
      RAISE NOTICE 'OK (Position_Code: computed)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:computed';
    END IF;

--  Verification of the x_y_position_code: Dead Reckoning
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'Dead Reckoning';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'Dead Reckoning Position' THEN
```

```
      RAISE NOTICE 'OK (Position_Code: Dead Reckoning)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:Dead Reckoning';
    END IF;

--  Verification of the x_y_position_code: LORAN
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'LORAN';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'LORAN C Fix' THEN
      RAISE NOTICE 'OK (Position_Code: LORAN)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:unknown';
    END IF;

--  Verification of the x_y_position_code: SAT
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'SAT';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'Transit Satellite Fix' THEN
      RAISE NOTICE 'OK (Position_Code: SAT)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:unknown';
    END IF;

--  Verification of the x_y_position_code: GPS
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'GPS';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'GPS position' THEN
      RAISE NOTICE 'OK (Position_Code: GPS)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:GPS';
    END IF;

--  Verification of the x_y_position_code: Waypoint
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'Waypoint';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =  'Waypoint Position' THEN
      RAISE NOTICE 'OK (Position_Code: Waypoint)';
    ELSE
      RAISE NOTICE 'Missmatch found in Position_Code:Waypoint';
    END IF;

--  Verification of the x_y_position_code: Interpolated
    SELECT description INTO tmp_rec FROM position_code WHERE x_y_position_code = 'Interpolated';
    IF tmp_rec IS NOT NULL AND tmp_rec.description =
      'Interpolated using two position fixes on either side of the resulting position values' THEN
      RAISE NOTICE 'OK (Position_Code: Interpolated)';
    ELSE
       RAISE NOTICE 'Missmatch found in Position_Code:Interpolated';
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyPositionCode() OWNER TO postgres;

-- Function which deletes the position_code table.
CREATE OR REPLACE FUNCTION deletePositionCode()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting position_code...';
    DELETE FROM position_code;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deletePositionCode() OWNER TO postgres;
```

Annex H: cruise_mngt_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

--  The following function fills the cruise, cruise_note and some of the data_package PDB tables
--  from the LDB database
CREATE OR REPLACE FUNCTION insertCruiseManagement()
RETURNS VOID AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_total_row_nb integer;
    tmp_row_index integer;
  BEGIN
--  ship table
    RAISE NOTICE 'Cruise Mngt: Inserting ship_name...';
--  Transfer of shipname from LDB to the ship table in the PDB
    INSERT INTO pdb.public.ship(ship_name) SELECT * FROM
      dblink('dbname=readbv3b user=postgres','SELECT shipname FROM
      readbv3b.public.ships WHERE shipname ~ ''[a-zA-Z1-9]''') AS t(a text);

--  cruise table
--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB cruise with a cursor at the beginning of the table
    PERFORM dblink_open('cruise_cursor', 'SELECT id  FROM readbv3b.public.cruises');

--  Calculating the total number of cruises (used in the next loop)
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*) FROM
      readbv3b.public.cruises ') AS t(b integer);
    tmp_total_row_nb = tmp_rec.b;

    RAISE NOTICE 'Cruise Mngt: Inserting cruise id...';

--  The following loop create a PDB data_package row entry (data_package_name and description)
--  for each cruise_id from the LDB. The data_package_id is equal to the cruise_id.
--  The data_package_name consist of the string "Cruise " followed by the cruise_id value.
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP
      SELECT * INTO tmp_rec FROM dblink_fetch('cruise_cursor',1) AS a(tmp_data_package_id integer);

      INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
        VALUES (tmp_rec.tmp_data_package_id,'Cruise '||tmp_rec.tmp_data_package_id,
        'Data package associate with the cruise number '|| tmp_rec.tmp_data_package_id);
    END LOOP;

--  Closing the cruise_cursor
    PERFORM dblink_close('cruise_cursor');

--  Transfer of cruises.id from LDB to the cruise.id column in the PDB
    INSERT INTO pdb.public.cruise(cruise_id) SELECT * FROM
      dblink('dbname=readbv3b user=postgres','SELECT id FROM readbv3b.public.cruises') AS t(a integer);
```

```
-- cruise_note table
    RAISE NOTICE 'Cruise Mngt: Inserting cruise note...';
--  Transfer of cruises.longdescription from LDB to the cruise_note.note column in the PDB
--  along with the cruise ID associated to it. Only non-empty descriptions are transfered.
    INSERT INTO pdb.public.cruise_note(cruise_id,note) SELECT * FROM
      dblink('dbname=readbv3b user=postgres','SELECT id,longdescription  FROM
      readbv3b.public.cruises WHERE longdescription ~ ''[a-zA-Z1-9]''')  AS t(a integer, b text);

-- scientist table
    RAISE NOTICE 'Cruise Mngt: Inserting scientist name...';
--  Transfer of scientists.scientist from LDB to the scientist.scientist_name column in the PDB
    INSERT INTO pdb.public.scientist(scientist_name) SELECT *
      FROM dblink('dbname=readbv3b user=postgres','SELECT scientist FROM readbv3b.public.scientists') AS t(a text);

-- scientist_on_cruise table
    RAISE NOTICE 'Cruise Mngt: Inserting scientist on cruise...';
--  Transfer of cruises.id_chief_scientist from LDB to the scientist_on_cruise.scientist_id column in the PDB
--  along with the cruise ID associated to it. Only non-empty entries are transfered.
--  The scientist ID is first read in the LDB than converted to its scientist name
--  than, using its name, the PDB scientist ID is found than transfered in the column scientist_on_cruise.scientist_id
    INSERT INTO pdb.public.scientist_on_cruise(cruise_id, scientist_id)
      SELECT a,getScientistIDFromName(getScientistNameFromID(b))
      FROM dblink('dbname=readbv3b user=postgres','SELECT id,id_chief_scientist FROM
      readbv3b.public.cruises WHERE id_chief_scientist > ''0''')
      AS t(a integer, b integer);
--  As this is the initial porting, the default value of scientist_counter was set to 1
--  in the alterations.sql file. We temporary the counter to 2 as we insert the second
--  scientist of the cruise.
    ALTER TABLE scientist_on_cruise ALTER COLUMN scientist_counter SET DEFAULT '2';

--  Transfer of cruises.id_chief_scientist_2 from LDB to the scientist_on_cruise.scientist_id column in the PDB
--  along with the cruise ID associated to it. Only non-empty and non-zero scientist_id are transfered.
--  The scientist ID is first read in the LDB than converted to its scientist name
--  than, using its name, the PDB scientist ID is found than transfered in the column scientist_on_cruise.scientist_id
    INSERT INTO pdb.public.scientist_on_cruise(cruise_id, scientist_id)
      SELECT a,getScientistIDFromName(getScientistNameFromID(b)) FROM
      dblink('dbname=readbv3b user=postgres','SELECT id,id_chief_scientist_2 FROM readbv3b.public.cruises
      WHERE id_chief_scientist_2 > ''0''')
      AS t(a integer, b integer);
--  We reset the default value to 1 (for new cruises)
--  This setting is only valid for the initial porting and not for updates. The last counter
--  value must be incremented instead.
    ALTER TABLE scientist_on_cruise ALTER COLUMN scientist_counter SET DEFAULT '1';

-- ships_on_cruise table
    RAISE NOTICE 'Cruise Mngt: Inserting ships ont cruise...';
--  Finding the ship_id associated with the only ship available in the LDB (Quest)
    SELECT ship_id INTO tmp_rec FROM ship WHERE ship_name = 'Quest';
--  In the LDB, there are no link between the ship name and the cruise
--  Fortunately, there is only one ship available (Quest) which is passed
--  as default to all the cruises id.
    INSERT INTO pdb.public.ships_on_cruise(cruise_id, ship_id) SELECT a,tmp_rec.ship_id
      FROM dblink('dbname=readbv3b user=postgres','SELECT id,id FROM readbv3b.public.cruises ')
      AS t(a integer, b integer);
  END;
$BODY$
   LANGUAGE 'plpgsql';
ALTER FUNCTION insertCruiseManagement() OWNER TO postgres;


CREATE OR REPLACE FUNCTION verifyCruiseManagement()
RETURNS VOID AS
$BODY$
  DECLARE
```

```
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Cruise Management ...';
    SELECT count(*) INTO tmp_rec FROM (SELECT ship_name FROM ship  EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT shipname FROM readbv3b.public.ships') AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (ship name)';
    ELSE
       RAISE NOTICE 'Missmatch found in ship' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT cruise_id FROM cruise  EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT id FROM readbv3b.public.cruises') AS t(b integer)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (cruise id)';
    ELSE
       RAISE NOTICE 'Missmatch found in cruise id' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT note FROM cruise_note  EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT longdescription FROM readbv3b.public.cruises')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (cruise note)';
    ELSE
       RAISE NOTICE 'Missmatch found in cruise_note(note)' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT scientist_name FROM scientist EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT scientist FROM readbv3b.public.scientists')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (scientist name)';
    ELSE
       RAISE NOTICE 'Missmatch found in scientist(scientist_name)' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT scientist_id FROM scientist_on_cruise EXCEPT
      SELECT scientist_id FROM scientist ) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (scientist_on_cruise scientist_id)';
    ELSE
       RAISE NOTICE 'Missmatch found in scientist_on_cruise(scientist_id)' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT cruise_id FROM scientist_on_cruise
      EXCEPT SELECT cruise_id FROM cruise ) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (scientist_on_cruise cruise_id)';
    ELSE
       RAISE NOTICE 'Missmatch found in scientist_on_cruise(cruise_id)' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT ship_id FROM ships_on_cruise
      EXCEPT SELECT ship_id FROM ship ) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (ships_on_cruise ship_id)';
    ELSE
       RAISE NOTICE 'Missmatch found in ships_on_cruise(ship_id)' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT cruise_id FROM ships_on_cruise
      EXCEPT ALL SELECT cruise_id FROM cruise ) AS t2(b2);
```

```
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (ships_on_cruise cruise_id)';
    ELSE
       RAISE NOTICE 'Missmatch found in ships_on_cruise(cruise_id)' ;
    END IF;

    SELECT count(*) INTO tmp_rec FROM (SELECT cruise_id FROM cruise
      EXCEPT SELECT data_package_id FROM data_package ) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (all cruise_id are found as data_package_id)';
    ELSE
       RAISE NOTICE 'Missmatch found: a cruise_id was not found as a data_package_id' ;
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyCruiseManagement() OWNER TO postgres;


CREATE OR REPLACE FUNCTION deleteCruiseManagement()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting ship_on_cruise...';
    DELETE FROM ships_on_cruise;
    RAISE NOTICE 'Deleting scientist_on_cruise...';
    DELETE FROM scientist_on_cruise;
    RAISE NOTICE 'Deleting scientist...';
    DELETE FROM scientist;
    RAISE NOTICE 'Deleting ship...';
    DELETE FROM ship;
    RAISE NOTICE 'Deleting cruise_note...';
    DELETE FROM cruise_note;
    RAISE NOTICE 'Deleting cruise...';
    DELETE FROM cruise;
    RAISE NOTICE 'Deleting data_package related to cruises...';
    DELETE FROM data_package WHERE data_package_id < 1000;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteCruiseManagement() OWNER TO postgres;
```

Annex I: xbtxsv_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertXBTXSV(input_year text, input_month text, input_day text)
RETURNS VOID AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_rec2 RECORD;
    tmp_rec3 RECORD;
    tmp_rec4 RECORD;
    tmp_row_index integer;
    tmp_row_index2 integer;
    tmp_total_row_nb integer;
    tmp_total_row_nb2  integer;
    tmp_featureid integer;
    tmp_stationid integer;
    tmp_sounding decimal;
    tmp_salinity decimal;
    tmp_coef_a decimal;
    tmp_coef_b decimal;
    tmp_surf_temp decimal;
    tmp_idfilename integer;
    tmp_geom_xyz geometry;
    tmp_measurement_location_id integer;
    tmp_measuring_device_id integer;
    tmp_long decimal;
    tmp_lat decimal;
    tmp_start_time timestamp without time zone;
    tmp_time_left varchar(200);
    tmp_xsv_soundspeed decimal(7,3);
    tmp_date_filter_str char(20);
    tmp_date_filter timestamp without time zone;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB xbt_file_meta_data with a cursor at the beginning of the table
    PERFORM dblink_open('xbt_cursor', 'SELECT id_filename, sequence_number, salinity_ppt, depth_m, depth_equation,
      depth_coefficient_1, depth_coefficient_2, depth_coefficient_3, probe_type, serial_number,
      surface_temperature_degc, terminal_depth_m, the_geom FROM readbv3b.public.xbt_file_meta_data');

-- Setting up the filter from the year,month,day passed as arguments
-- A very low default value is set 01 January 1900 is no inputs are given
-- A timestamp is generated from the inputs. All three inputs must be numeric
-- for the filter to be activated. Later in this file, the filter (tmp_date_filter)
-- is compared to LDB date value.
    tmp_date_filter_str = '1900'||'01'||'01'||'00'||'00'||'00';
    tmp_date_filter = to_timestamp(tmp_date_filter_str,'YYYYMMDDHHMISS');
```

```
    IF input_year ~ '[0-9]' AND input_month ~ '[0-9]' AND input_day ~ '[0-9]'  THEN
      tmp_date_filter_str = input_year||input_month||input_day||'00'||'00'||'00';
      tmp_date_filter = to_timestamp(tmp_date_filter_str,'YYYYMMDDHHMISS');
    END IF;

-- This command retrieves the total number of row of the xbt_file_meta_data,
-- this number is used in the following loop.
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*)
      FROM readbv3b.public.xbt_file_meta_data') AS a(total_row_nb integer);
    tmp_total_row_nb = tmp_rec.total_row_nb;

-- Looping on all xbt_file_meta_data entries
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP
-- Initialization of local variable on each loop iteration for test checking
      tmp_featureid = NULL;
      tmp_stationid = NULL;
      tmp_sounding = NULL;
      tmp_salinity = NULL;
      tmp_coef_a = NULL;
      tmp_coef_b = NULL;
      tmp_surf_temp = NULL;
      tmp_idfilename = NULL;

-- Standard script for measuring and printing the time for executing this script
      IF tmp_row_index % 20 = 0 THEN
        tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
        RAISE NOTICE 'XBT/XSV %', tmp_time_left;
      END IF;

-- Fetching one row of the xbt_file_meta_data table and storing the result in tmp_rec
      SELECT * INTO tmp_rec FROM dblink_fetch('xbt_cursor',1) AS a(id_filename integer, sequence_number text,
        salinity_ppt text, depth_m text, depth_equation text, depth_coefficient_1 text, depth_coefficient_2 text,
        depth_coefficient_3 text, probe_type text, serial_number text, surface_temperature_degc text,
        terminal_depth_m text, the_geom geometry);

-- Using the id_filename from above, we retrieve the timestamp and the geometry associated with it from the geopoints table
      SELECT * INTO tmp_rec2 FROM dblink('dbname=readbv3b user=postgres','SELECT time,the_geom FROM readbv3b.public.geopoints
        WHERE id_filename ='|| tmp_rec.id_filename) AS b(time timestamp  without time zone, the_geom geometry);

-- Application of the timestamp filter, if the XBT/XSV time is lower than the filter, the entry and the following
-- processing is skipped.
      IF tmp_rec2.time > tmp_date_filter THEN

-- Because the measuring device set by the LDB do not fit exactly with the PDB device layout,
-- a function was created to properly assigned the PDB devices from the probe_type (see that function for details)
        tmp_measuring_device_id = getMeasuringDeviceIDFromName(tmp_rec.probe_type);

        tmp_idfilename = tmp_rec.id_filename;

-- Converting strings containing numerals into their equivalent numeric version (integer or decimal)
-- the long format sequence, second argument of the function to_number, is necessary for retrieving
-- all the digits from the strings.
        IF  tmp_rec.sequence_number ~ '[0-9]' THEN
          tmp_stationid := to_number(tmp_rec.sequence_number,'9999999');
        END IF;
        IF  tmp_rec.salinity_ppt ~ '[0-9]' THEN
          tmp_salinity := to_number(tmp_rec.salinity_ppt,'9999.99');
        END IF;
        IF  tmp_rec.depth_m ~ '[0-9]' THEN
          tmp_sounding := to_number(tmp_rec.depth_m,'999999.99');
        END IF;
        IF  tmp_rec.surface_temperature_degc ~ '[0-9]' THEN
          tmp_surf_temp := to_number(tmp_rec.surface_temperature_degc,'999.99');
        END IF;
```

```
-- The following lines, associating coefficients 1,2 and 3 to the two coefficients a and b
-- follow the rules defined in the design report. Usually coef 1 and 2 are assigned to A and B.
-- If coefficient 1 is NULL or undefined, coefficient 2 and 3 are assigned to A and B respectively
        IF tmp_rec.depth_coefficient_1 ~ '[1-9]' THEN
          tmp_coef_a := to_number(tmp_rec.depth_coefficient_1,'S999999.99999999');
          IF tmp_rec.depth_coefficient_2 ~ '[1-9]' THEN
            tmp_coef_b := to_number(tmp_rec.depth_coefficient_2,'S999999.99999999');
          END IF;
        ELSE
          IF tmp_rec.depth_coefficient_2 ~ '[1-9]' THEN
            tmp_coef_a := to_number(tmp_rec.depth_coefficient_2,'S999999.99999999');
          END IF;
          IF tmp_rec.depth_coefficient_3 ~ '[1-9]' THEN
            tmp_coef_b := to_number(tmp_rec.depth_coefficient_3,'S999999.99999999');
          END IF;
        END IF;
--  Creating a feature_id associated to this data package. Note that the data_package_id
--  is defined in this case as the cruise_id
        tmp_featureid = createFeature('IP',getCruiseIDFromIDFilename(tmp_rec.id_filename));

-- Creation of an instantaneous point entry
        INSERT INTO pdb.public.instantaneous_point(feature_id, feature_code, cruise_id, station_id,
          point_type, date_time)
          VALUES ( tmp_featureid, 'XBT', getCruiseIDFromIDFilename(tmp_rec.id_filename), tmp_stationid,
          1, tmp_rec2.time);

-- Creation of an PDB xbt_xsv entry for each LDB xbt_file_meta_data entry
        INSERT INTO pdb.public.xbt_xsv(feature_id, assumed_salinity, surface_temperature, sounding,
          equation, coefficient_a, coefficient_b, serial_number)
          VALUES ( tmp_featureid , tmp_salinity, tmp_surf_temp, tmp_sounding, tmp_rec.depth_equation,
          tmp_coef_a, tmp_coef_b,tmp_rec.serial_number);

-- For each LSB xbt_file_meta_data entry, we use the id_filename for selecting all the data
-- contained in the LDB xbt_profiles related to that id_filename. First we count the number of
-- entries of the subset of data related to the id_filename. We then open a cursor in that
-- data subset.
        SELECT * INTO tmp_rec3 FROM dblink('dbname=readbv3b user=postgres','SELECT count(*)
          FROM readbv3b.public.xbt_profiles
          WHERE id_filename ='|| tmp_idfilename ) AS a(total_row_nb integer);
--      The casting is to preserved the decimals hidden in the LDB version.
        PERFORM dblink_open('xbt_profiles_cursor', 'SELECT depth_m,CAST(soundspeed_mps AS double precision),temperature_degc
          FROM readbv3b.public.xbt_profiles WHERE id_filename ='|| tmp_idfilename);
        tmp_total_row_nb2 = tmp_rec3.total_row_nb;
        -- Debugging lines
        -- RAISE NOTICE 'DEBUG %', tmp_row_index||' '||tmp_total_row_nb||' '||tmp_total_row_nb2
        -- ||' '||tmp_idfilename;
-- Looping the rows of the data subset specific to a given LDB id_filename
        FOR tmp_row_index2 IN 1..tmp_total_row_nb2 LOOP

-- Fetching one row (soundspeed and water temperature)of the data subset and storing the result in tmp_rec3
          SELECT * INTO tmp_rec3 FROM  dblink_fetch('xbt_profiles_cursor',1) AS a2(depth_m real,
            soundspeed_mps double precision, temperature_degc real);

-- A new 3D geometry is constructed from the old 2D geometry and the LDB deptm_m value
          tmp_long = ST_X(tmp_rec2.the_geom);
          tmp_lat = ST_Y(tmp_rec2.the_geom);
          tmp_geom_xyz = GeomFromEWKT('SRID=4326;POINT('||tmp_long||' '|| tmp_lat||' '|| tmp_rec3.depth_m ||')');
-- The casting is to preserved the decimals hidden in the LDB version.
          tmp_xsv_soundspeed = CAST( CAST(tmp_rec3.soundspeed_mps AS DOUBLE PRECISION) AS DECIMAL(7,3));

-- Creation of a measurement location entry using the new 3D geometry
          INSERT INTO pdb.public.measurement_location(feature_id, feature_code, geom, x_y_position_code)
            VALUES (tmp_featureid,'XBT',tmp_geom_xyz, 'unknown') RETURNING measurement_location_id
```

```
            INTO tmp_measurement_location_id;

-- Creating a data PDB entry for the temperature and soundspeed. We skip null entries.
-- If the soundspeed is not null and the probe_type is either XBTT5, XBTT7 or SBE19PLUS, then
-- the measuring device is "1" (computed). For the other devices, then their ids is used to
-- identify the measuring_device associated with the measurement.
          IF tmp_rec3.temperature_degc IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,322,tmp_measuring_device_id,1,tmp_rec3.temperature_degc,0);
            IF tmp_rec3.soundspeed_mps IS NOT NULL THEN
              IF tmp_rec.probe_type ~ 'T-5' OR tmp_rec.probe_type ~ 'T-7' OR tmp_rec.probe_type ~ 'SBE19' THEN
                INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
                  group_id, data_value, quality_flag_id)
                  VALUES (tmp_measurement_location_id,617,1,1,tmp_rec3.soundspeed_mps,0);
              ELSE
                INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
                  group_id, data_value, quality_flag_id)
                  VALUES (tmp_measurement_location_id,617,tmp_measuring_device_id,1,tmp_rec3.soundspeed_mps,0);
              END IF;
            END IF;
          ELSE
-- If the temperature is null but not the soundspeed, then all soundspeed data are measured (not computed)
-- and the probe related measuring_device_id is used to identify the device that measure the soundspeed.
            IF tmp_rec3.soundspeed_mps IS NOT NULL THEN
              INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
                group_id, data_value, quality_flag_id)
                VALUES (tmp_measurement_location_id,617,tmp_measuring_device_id,1,tmp_xsv_soundspeed,0);
            END IF;
          END IF;
        END LOOP;
        PERFORM dblink_close('xbt_profiles_cursor');
      END IF;
    END LOOP;
    PERFORM dblink_close('xbt_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertXBTXSV(text,text,text) OWNER TO postgres;

-- Function that verifies that XBT/XSV data are in the PDB
CREATE OR REPLACE FUNCTION verifyXBTXSV()
RETURNS VOID AS
$BODY$
  DECLARE
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying XBT/XSV data...';
--    This verification command was commented as it was not valid anymore due to additional
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.
--    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 617
--      EXCEPT ALL SELECT b FROM
--      dblink('dbname=readbv3b user=postgres','SELECT soundspeed_mps FROM readbv3b.public.xbt_profiles')
--      AS t(b double precision)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (XBT/XSV soundspeed data)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in XBT/XSV data(data_value), parameter_id = 617';
--    END IF;

-- Verification of the XBT/XSV data value (parameter_id = 322) temperature_degc
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 322
      EXCEPT ALL SELECT b FROM
```

```
      dblink('dbname=readbv3b user=postgres','SELECT temperature_degc FROM readbv3b.public.xbt_profiles')
      AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (XBT/XSV temperature data)';
    ELSE
       RAISE NOTICE 'Missmatch found in XBT/XSV data(data_value), parameter_id = 322';
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyXBTXSV() OWNER TO postgres;

-- Function which deletes the xbt_xsv table and related data.
CREATE OR REPLACE FUNCTION deleteXBTXSV()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting XBT/XSV data (temperature)...';
    DELETE FROM data WHERE parameter_id = 322;
    RAISE NOTICE 'Deleting XBT/XSV data (soundspeed)...';
    DELETE FROM data WHERE parameter_id = 617;
    RAISE NOTICE 'Deleting measurement location related to XBT/XSV...';
    DELETE FROM measurement_location WHERE feature_code = 'XBT';
    RAISE NOTICE 'Deleting xbt_xsv...';
    DELETE FROM xbt_xsv;
    RAISE NOTICE 'Deleting instantaneous_point related to XBT/XSV...';
    DELETE FROM instantaneous_point WHERE feature_code = 'XBT';
    RAISE NOTICE 'Done';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteXBTXSV() OWNER TO postgres;
```

This page intentionally left blank.

Annex J: areas_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertAreas()
RETURNS VOID AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_rec2 RECORD;
    tmp_total_row_nb integer;
    tmp_featureid integer;
    tmp_start_time timestamp without time zone;
    tmp_time_left varchar(200);
    tmp_data_package_id integer;
    tmp_geom geometry;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB opareas with a cursor at the beginning of the table
    PERFORM dblink_open('opareas_cursor', 'SELECT id, sea_areas, air_space FROM readbv3b.public.opareas');

--  Creation of the data package ID and insertion of a new entry in the data_package table
    tmp_data_package_id = 1000001;
    INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
      VALUES (tmp_data_package_id,'OPAREAS','Data package from the opareas data FROM READBV2B');

--  Evaluating the total number of rows in the table opareas
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*) FROM readbv3b.public.opareas ')
      AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;

--  The following loop processes all the OPAREA data
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

--  Code for measuring and printing the time for executing this script
      tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
      RAISE NOTICE 'AREAS: %', tmp_time_left;

--  Fetching the next row to be process in the LDB table opareas
      SELECT * INTO tmp_rec FROM dblink_fetch('opareas_cursor',1)
        AS a(oparea_id integer, sea_areas text, air_space text);

--  We skip the line associated to ROMEO (Halifax Harbour) because
--  no geometry was found for this entry. Only entries with bounds = TRUE are processed.
      IF tmp_rec.sea_areas <> 'ROMEO' THEN
        SELECT * INTO tmp_rec2 FROM dblink('dbname=readbv3b user=postgres','SELECT the_geom FROM
          readbv3b.public.opareas_areas WHERE bounds IS TRUE AND id_oparea ='||
          tmp_rec.oparea_id) AS a(the_geom geometry);
```

```
--  Creating a feature_id associated to this data package
       tmp_featureid = createFeature('FA',tmp_data_package_id);
--  The SRID is ported from 4269 to 4326.
       tmp_geom = ST_Transform(st_multi(tmp_rec2.the_geom), 4326);

--  Creating a new PDB entry for the table feature_area
       INSERT INTO pdb.public.feature_area(feature_id, area_name, feature_code, geom)
         VALUES (tmp_featureid,'OPAREA',tmp_rec.sea_areas, tmp_geom);
       IF tmp_rec.air_space IS NOT NULL THEN
--  Creating a new PDB entry for the table area_characteristic
          INSERT INTO pdb.public.area_characteristic(feature_id, name, value)
            VALUES (tmp_featureid,'AirSpaceHeightLimit',tmp_rec.air_space);
       END IF;
     END IF;
   END LOOP;
   PERFORM dblink_close('opareas_cursor');

--  Creation of the data package ID and insertion of a new entry in the data_package table
   tmp_data_package_id = 1000002;
   INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
     VALUES (1000002,'MARLOA','Data package from the marloa data FROM READBV2B');

--  opening a connection to the LDB marloa data with a cursor at the beginning of the table
   PERFORM dblink_open('marloa_cursor', 'SELECT sea_areas, creator, department, country, date,
     the_geom FROM readbv3b.public.marloa');

--  Evaluating the total number of rows in the table marloa
   SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*) FROM
     readbv3b.public.marloa ') AS a(b integer);
   tmp_total_row_nb = tmp_rec.b;

--  The following loop processes all the MARLOA data
   FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

--  Code for measuring and printing the time for executing this script
     tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
     RAISE NOTICE 'AREAS: %', tmp_time_left;

-- Fetching the next row to be process in the LDB table marloa
     SELECT * INTO tmp_rec FROM dblink_fetch('marloa_cursor',1) AS a(sea_areas text, creator text,
       department text, country text, date text, the_geom geometry);

--  Creating a feature_id associated to this data package
     tmp_featureid = createFeature('FA',tmp_data_package_id);
--  The SRID is ported from 4269 to 4326.
     tmp_geom = ST_Transform(tmp_rec.the_geom, 4326);

--  Creating a new PDB entry for the table feature_area
     INSERT INTO pdb.public.feature_area(feature_id, area_name, feature_code, geom)
       VALUES (tmp_featureid,'MARLOA',tmp_rec.sea_areas,tmp_geom);
     IF tmp_rec.creator IS NOT NULL THEN
       INSERT INTO pdb.public.area_characteristic(feature_id, name, value)
         VALUES (tmp_featureid,'creator',tmp_rec.creator);
     END IF;
     IF tmp_rec.department IS NOT NULL THEN
--  Creating a new PDB entry for the table area_characteristic for column department
       INSERT INTO pdb.public.area_characteristic(feature_id, name, value)
         VALUES (tmp_featureid,'department',tmp_rec.department);
     END IF;
     IF tmp_rec.country IS NOT NULL THEN
--  Creating a new PDB entry for the table area_characteristic for column country
       INSERT INTO pdb.public.area_characteristic(feature_id, name, value)
         VALUES (tmp_featureid,'country',tmp_rec.country);
```

```
      END IF;
      IF tmp_rec.date IS NOT NULL THEN
--  Creating a new PDB entry for the table area_characteristic for column date
        INSERT INTO pdb.public.area_characteristic(feature_id, name, value)
          VALUES (tmp_featureid,'date',tmp_rec.date);
      END IF;
    END LOOP;
    PERFORM dblink_close('marloa_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertAreas() OWNER TO postgres;

-- Function that verifies that the areas are in place in the PDB
CREATE OR REPLACE FUNCTION verifyAreas()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Areas...';
--  Verification of the feature_code in feature_area OPAREA
    SELECT count(*) INTO tmp_rec FROM (SELECT feature_code FROM feature_area
      WHERE area_name = 'OPAREA' EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT sea_areas FROM readbv3b.public.opareas')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area opareas data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area data(data_value), area_name = OPAREA';
    END IF;

--    SELECT count(*) INTO tmp_rec FROM (SELECT geom FROM feature_area WHERE area_name = 'OPAREA'
--      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres',
--     'SELECT the_geom FROM readbv3b.public.opareas_areas WHERE bounds IS TRUE AND id_oparea <> 43')
--      AS t(b geometry)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (Area opareas geom data)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in Area data(data_value), area_name = OPAREA';
--    END IF;

--  Verification of the area_characteristic values for AirSpaceHeightLimit
    SELECT count(*) INTO tmp_rec FROM (SELECT value FROM area_characteristic WHERE name = 'AirSpaceHeightLimit'
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT air_space FROM readbv3b.public.opareas')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area opareas air_space data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area Characteristic value, name = AirSpaceHeightLimit';
    END IF;

--  Verification of the feature_code in feature_area MARLOA
    SELECT count(*) INTO tmp_rec FROM (SELECT feature_code FROM feature_area WHERE area_name = 'MARLOA'
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT sea_areas FROM readbv3b.public.marloa')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area marloa data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area data(data_value), area_name = MARLOA';
    END IF;

--    SELECT count(*) INTO tmp_rec FROM (SELECT geom FROM feature_area WHERE area_name = 'MARLOA'
```

```
--      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT the_geom FROM readbv3b.public.marloa')
--      AS t(b geometry)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (Area Marloa geom data)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in Area data(data_value), area_name = OPAREA';
--    END IF;

--  Verification of the area_characteristic values for creator
    SELECT count(*) INTO tmp_rec FROM (SELECT value FROM area_characteristic WHERE name = 'creator'
      EXCEPT ALL SELECT b FROM  dblink('dbname=readbv3b user=postgres','SELECT creator FROM readbv3b.public.marloa')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area marloa creator data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area Characteristic value, name = creator';
    END IF;

--  Verification of the area_characteristic values for department
    SELECT count(*) INTO tmp_rec FROM (SELECT value FROM area_characteristic WHERE name = 'department'
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT department FROM readbv3b.public.marloa')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area marloa department data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area Characteristic value, name = department';
    END IF;

--  Verification of the area_characteristic values for country
    SELECT count(*) INTO tmp_rec FROM (SELECT value FROM area_characteristic WHERE name = 'country'
      EXCEPT ALL SELECT b FROM  dblink('dbname=readbv3b user=postgres','SELECT country FROM readbv3b.public.marloa')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area marloa country data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area Characteristic value, name = country';
    END IF;

--  Verification of the area_characteristic values for date
    SELECT count(*) INTO tmp_rec FROM (SELECT value FROM area_characteristic WHERE name = 'date'
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT date FROM readbv3b.public.marloa')
      AS t(b text)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Area marloa date data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Area Characteristic value, name = date';
    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyAreas() OWNER TO postgres;

-- Function which deletes the feature_area and area_characteristic table.
CREATE OR REPLACE FUNCTION deleteAreas()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting area_characteristic...';
    DELETE FROM area_characteristic;
    RAISE NOTICE 'Deleting feature_area...';
    DELETE FROM feature_area;
    RAISE NOTICE 'Deleting feature related to areas...';
    DELETE FROM feature WHERE data_package_id = 1000001;
```

```
    DELETE FROM feature WHERE data_package_id = 1000002;
    RAISE NOTICE 'Deleting data_package related to areas...';
    DELETE FROM data_package WHERE data_package_id = 1000001;
    DELETE FROM data_package WHERE data_package_id = 1000002;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteAreas() OWNER TO postgres;
```

This page intentionally left blank.

Annex K: tsd_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertTSD()
RETURNS VOID AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_rec2 RECORD;
    tmp_rec3 RECORD;
    tmp_row_index integer;
    tmp_total_row_nb  integer;
    tmp_month_index integer;
    tmp_index integer;
    tmp_date_time timestamp without time zone;
    tmp_featureid integer[];
    tmp_geom_xyz geometry[];
    tmp_long decimal;
    tmp_lat decimal;
    tmp_start_time timestamp without time zone;
    tmp_time_left varchar(200);
    tmp_data_package_id integer;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Creation of the data package ID and insertion of a new entry in the data_package table
    tmp_data_package_id = 3000001;
    INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
      VALUES (tmp_data_package_id,'TSD','Data package associated to the Dalhousie Temperature Salinity dataset');

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB tsd_geopoints with a cursor at the beginning of the table
    PERFORM dblink_open('tsd_cursor', 'SELECT pid, depth, the_geom FROM readbv3b.public.tsd_geopoints');

-- Creation of a mesh_source entry
    INSERT INTO pdb.public.mesh_source(source,description) VALUES  ('Dalhousie TSD',
      'Dalhousie Temperature Salinity data set');
-- Creation of a mesh entry
    INSERT INTO pdb.public.mesh(source) VALUES ('Dalhousie TSD');

-- Counting the total number of rows of tsd_geopoints, needed for looping on the table
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres',
      'SELECT count(*) FROM readbv3b.public.tsd_geopoints ') AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;

-- Looping on all tsd_geopoints entries
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

-- Standard script for measuring and printing the time for executing this script
```

```
      IF tmp_row_index % 400 = 0 THEN
        tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
        RAISE NOTICE 'TSD: %', tmp_time_left;
      END IF;

-- Fetching one row of the tsd_geopoints table and storing the result in tmp_rec
      SELECT * INTO tmp_rec FROM dblink_fetch('tsd_cursor',1) AS a(pid integer, depth double precision[],
        the_geom geometry);

-- Looping on the months
      FOR tmp_month_index IN 1..12 LOOP

-- Setting the timestamp for this dataset, bogus year and time are used (00),
-- only the month is relevant
        IF tmp_month_index < 10 THEN
          tmp_date_time = to_timestamp('000'||tmp_month_index||'01000000','YYMMDDHHMISS');
        ELSE
          tmp_date_time = to_timestamp('00'||tmp_month_index||'01000000','YYMMDDHHMISS');
        END IF;

--  Opening a connection to the LDB tsd_salinity table with a cursor pointing to the subset associated to a
--  specific month and a specific PID from the tsd_geopoints table
        PERFORM dblink_open('tsd_salinity_cursor', 'SELECT sal  FROM readbv3b.public.tsd_salinity WHERE PID = '
          ||tmp_rec.pid||'AND month = '||tmp_month_index);

--  Opening a connection to the LDB tsd_temperature table with a cursor pointing to the subset associated to a
--  specific month and a specific PID from the tsd_geopoints table
        PERFORM dblink_open('tsd_temperature_cursor','SELECT temp FROM readbv3b.public.tsd_temperature WHERE PID = '
          ||tmp_rec.pid||'AND month = '||tmp_month_index );

-- Fetching one row of the tsd_salinity table for a given month and a PID
        SELECT * INTO tmp_rec2 FROM dblink_fetch('tsd_salinity_cursor',1) AS a(sal double precision[]);
-- Fetching one row of the tsd_temperature table for a given month and a PID
        SELECT * INTO tmp_rec3 FROM dblink_fetch('tsd_temperature_cursor',1) AS b(temp double precision[]);

-- If the value for salinity is higher than zero, proceed with the porting
        IF trunc(tmp_rec2.sal[1]) != 0 THEN

-- Looping over 15 values of the arrays
          FOR tmp_index IN 1..15 LOOP

-- For the first month, we set the 3D geometry that will be used in all the others
            IF tmp_month_index = 1 THEN
              tmp_long = ST_X(tmp_rec.the_geom);
              tmp_lat = ST_Y(tmp_rec.the_geom);
              tmp_geom_xyz[tmp_index] = GeomFromEWKT('SRID=4326;POINT('||tmp_long||' '|| tmp_lat||' '||
                abs(tmp_rec.depth[tmp_index]) ||')');

--  Creating a feature_id associated to this data package
              tmp_featureid[tmp_index] = createFeature('IP',tmp_data_package_id);
-- Creating a mesh_point entry with the 3D geometry value
              INSERT INTO pdb.public.mesh_point(feature_id, source, geom, mesh_point_type)
                VALUES (tmp_featureid[tmp_index], 'Dalhousie TSD', tmp_geom_xyz[tmp_index], 'GridPoint');
            END IF;
-- Creating a scalar_quantity value for the salinity (parameter id = 402)
            INSERT INTO pdb.public.scalar_quantity(feature_id, parameter_id, date_time, data_value)
              VALUES (tmp_featureid[tmp_index], '402', tmp_date_time, tmp_rec2.sal[tmp_index]);
-- Creating a scalar_quantity value for the temperature (parameter id = 322)
            INSERT INTO pdb.public.scalar_quantity(feature_id, parameter_id, date_time, data_value)
              VALUES (tmp_featureid[tmp_index], '322', tmp_date_time, tmp_rec3.temp[tmp_index]);
          END LOOP;
        END IF;
        PERFORM dblink_close('tsd_temperature_cursor');
        PERFORM dblink_close('tsd_salinity_cursor');
```

```
      END LOOP;
    END LOOP;
  PERFORM dblink_close('tsd_cursor');
  END;
$BODY$
   LANGUAGE 'plpgsql';
ALTER FUNCTION insertTSD() OWNER TO postgres;

-- Function that verifies that TSD data are in the PDB
CREATE OR REPLACE FUNCTION verifyTSD()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying TSD data...';
--    This verification command was commented as it was not valid anymore due to additional
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.
--    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM scalar_quantity WHERE parameter_id = 322
--      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT temp
--      FROM readbv3b.public.tsd_temperature') AS t(b real)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (TSD temperature data)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in TSD scalar_quantity(data_value), parameter_id = 322';
--    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyTSD() OWNER TO postgres;

-- Function which deletes the data related to TSD.
CREATE OR REPLACE FUNCTION deleteTSD()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting scalar_quantity related to TSD (salinity)...';
    DELETE FROM scalar_quantity WHERE parameter_id = 402;
    RAISE NOTICE 'Deleting scalar_quantity related to TSD (temperature)...';
    DELETE FROM scalar_quantity WHERE parameter_id = 322;
    RAISE NOTICE 'Deleting mesh_point related to TSD...';
    DELETE FROM mesh_point WHERE source = 'Dalhousie TSD';
    RAISE NOTICE 'Deleting mesh related to TSD...';
    DELETE FROM mesh WHERE source = 'Dalhousie TSD';
    RAISE NOTICE 'Deleting mesh_source related to TSD...';
    DELETE FROM mesh_source WHERE source = 'Dalhousie TSD';
    RAISE NOTICE 'Deleting data_package_id related to TSD...';
    DELETE FROM data_package WHERE data_package_id = 3000001;
    RAISE NOTICE 'Done';
  END;
$BODY$
   LANGUAGE 'plpgsql';
ALTER FUNCTION deleteTSD() OWNER TO postgres;
```

This page intentionally left blank.

Annex L: sedthick_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertSedThick()
RETURNS VOID AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_row_index integer;
    tmp_total_row_nb  integer;
    tmp_date_time timestamp;
    tmp_featureid integer;
    tmp_start_time timestamp;
    tmp_time_left varchar(200);
    tmp_data_package_id integer;
    tmp_geom geometry;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Creation of the data package ID and insertion of a new entry in the data_package table
    tmp_data_package_id = 4000001;
    INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
      VALUES (tmp_data_package_id,'SedThick','Data package associated to Sediment Thickness dataset');

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB sedthick with a cursor at the beginning of the table
    PERFORM dblink_open('sedthick_cursor', 'SELECT the_geom,thickness FROM readbv3b.public.sedthick');

-- Creation of a mesh_source entry
    INSERT INTO pdb.public.mesh_source(source,description) VALUES
      ('NGDC SedThick','NGDC Total Sediment Thickness of the World Oceans & Marginal Seas (online)');
-- Creation of a mesh entry
    INSERT INTO pdb.public.mesh(source) VALUES ('NGDC SedThick');

-- Counting the total number of rows of sedthick, needed for looping on the table
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*)
      FROM readbv3b.public.sedthick ') AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;

-- Setting the timestamp for this dataset, as found in the design document (or following discussions)
    tmp_date_time = to_timestamp('060724155900','YYMMDDHHMISS');

-- Standard script for measuring and printing the time for executing this script
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP
      IF tmp_row_index % 10000 = 0 THEN
        tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
        RAISE NOTICE 'Sed.Thick:%', tmp_time_left;
      END IF;
```

```
-- Fetching one row of the sedthick table and storing the result in tmp_rec
      SELECT * INTO tmp_rec FROM dblink_fetch('sedthick_cursor',1) AS a(the_geom geometry, thickness real);

--  Creating a feature_id associated to this data package
      tmp_featureid = createFeature('IP',tmp_data_package_id);

--  The SRID is ported from 4269 to 4326.
      tmp_geom = ST_Transform(tmp_rec.the_geom,4326);

-- Creation of a mesh_point entry
      INSERT INTO pdb.public.mesh_point(feature_id, source, geom, mesh_point_type)
        VALUES (tmp_featureid, 'NGDC SedThick', tmp_geom, 'GridPoint');

-- Creation of a scalar_quantity entry
      INSERT INTO pdb.public.scalar_quantity(feature_id, parameter_id, date_time, data_value)
        VALUES (tmp_featureid, 401, tmp_date_time, tmp_rec.thickness);

    END LOOP;
    PERFORM dblink_close('sedthick_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertSedThick() OWNER TO postgres;

-- Function that verifies that SedThick data are in the PDB
CREATE OR REPLACE FUNCTION verifySedThick()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Sediment Thickness data...';
-- Verification of the SedThick data value (parameter_id = 401 ) thickness
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM scalar_quantity WHERE parameter_id = 401
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT thickness
      FROM readbv3b.public.sedthick') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sed.Thick. thickness data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sed.Thick. scalar_quantity(data_value), parameter_id = 401';
    END IF;

--    This verification command was commented as it was not valid anymore due to additional
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.
--    SELECT count(*) INTO tmp_rec FROM (SELECT geom FROM mesh_point WHERE source = 'NGDC SedThick'
--      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT the_geom
--      FROM readbv3b.public.sedthick') AS t(b geometry)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (Sed.Thick. geom)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in Sed.Thick. mesh_point(geom)';
--    END IF;
--    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifySedThick() OWNER TO postgres;

-- Function which deletes the data related to SedThick.
CREATE OR REPLACE FUNCTION deleteSedThick()
RETURNS VOID AS
$BODY$
```

```
  BEGIN
    RAISE NOTICE 'Deleting scalar_quantity related to Sediment Thickness data...';
    DELETE FROM scalar_quantity WHERE parameter_id = 401;
    RAISE NOTICE 'Deleting mesh_point related to Sediment Thickness data...';
    DELETE FROM mesh_point WHERE source = 'NGDC SedThick';
    RAISE NOTICE 'Deleting mesh related to Sediment Thickness data...';
    DELETE FROM mesh WHERE source = 'NGDC SedThick';
    RAISE NOTICE 'Deleting mesh_source related to Sediment Thickness data...';
    DELETE FROM mesh_source WHERE source = 'NGDC SedThick';
    RAISE NOTICE 'Deleting data_package_id related to Sediment Thickness data...';
    DELETE FROM data_package WHERE data_package_id = 4000001;
    RAISE NOTICE 'Done.';
  END;
$BODY$
   LANGUAGE 'plpgsql';
ALTER FUNCTION insertSedThick() OWNER TO postgres;
```

This page intentionally left blank.

Annex M: nadas_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertNADAS(input_year text, input_month text, input_day text)
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_rec2 RECORD;
    tmp_rec3 RECORD;
    tmp_row_index integer;
    tmp_total_row_nb  integer;
    tmp_featureid integer;
    tmp_measurement_location_id integer;
    tmp_start_time timestamp without time zone;
    tmp_time_left varchar(200);
    tmp_geom geometry;
    tmp_date_filter_str char(20);
    tmp_date_filter timestamp without time zone;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB nadas_observations with a cursor at the beginning of the table
    PERFORM dblink_open('nadas_cursor', 'SELECT air_temperature_degc, barometric_pressure_mbar,
      course_over_ground_degt, depth_m, percipitation_mm, relative_humidity_pct, ship_heading_degt,
      ship_propeller_port_rpm, speed_over_ground_kt, surface_water_temperature_degc, time, id_filename,
      wind_speed_kt, wind_dir_degt, visibility_solar, ship_propeller_stbd_rpm, the_geom, the_geom_drdccode
      FROM readbv3b.public.nadas_observations');

-- Setting up the filter from the year,month,day passed as arguments
-- A very low default value is set 01 January 1900 is no inputs are given
-- A timestamp is generated from the inputs. All three inputs must be numeric,
-- for the filter to be activated. The year must 4 digits, and 2 digits
-- is needed for the month and day, using 0 is necessary. Later in this file,
-- the filter (tmp_date_filter) is compared to LDB date value.
    tmp_date_filter_str = '1900'||'01'||'01'||'00'||'00'||'00';
    tmp_date_filter = to_timestamp(tmp_date_filter_str,'YYYYMMDDHHMISS');
    IF input_year ~ '[0-9]' AND input_month ~ '[0-9]' AND input_day ~ '[0-9]'  THEN
      tmp_date_filter_str = input_year||input_month||input_day||'00'||'00'||'00';
      tmp_date_filter = to_timestamp(tmp_date_filter_str,'YYYYMMDDHHMISS');
    END IF;

-- Counting the total number of rows of nadas_observations, needed for looping on the table
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*)
      FROM readbv3b.public.nadas_observations ') AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;

-- Looping on all xbt_file_meta_data entries
```

```
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

-- Standard script for measuring and printing the time for executing this script
      IF tmp_row_index % 10000 = 0 THEN
        tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
        RAISE NOTICE 'NADAS: %', tmp_time_left;
      END IF;

-- Fetching one row of the nadas_observations table and storing the result in tmp_rec
      SELECT * INTO tmp_rec FROM dblink_fetch('nadas_cursor',1) AS a1(air_temperature_degc real,
        barometric_pressure_mbar real, course_over_ground_degt real, depth_m real, percipitation_mm real,
        relative_humidity_pct real, ship_heading_degt real, ship_propeller_port_rpm real, speed_over_ground_kt real,
        surface_water_temperature_degc real, time timestamp without time zone, id_filename integer, wind_speed_kt real,
        wind_dir_degt real,  visibility_solar real, ship_propeller_stbd_rpm real, the_geom geometry,
        the_geom_drdccode integer);

-- Application of the timestamp filter, if the NADAS time is lower than the filter, the entry and the following
-- processing is skipped.
      IF tmp_rec.time > tmp_date_filter THEN

-- Using the the_geom_drdccode from the nadas_observations table, we fetch the associated shortdescription
-- and put it in a xy_code_desc variable
        SELECT * INTO tmp_rec2 FROM dblink('dbname=readbv3b user=postgres','SELECT shortdescription
          FROM readbv3b.public.drdctypes WHERE id = '|| tmp_rec.the_geom_drdccode) AS a2(xy_code_desc text);
-- Using the above xy_code_desc, we fetch its equivalent in the PDB x_y_position_code by comparing
-- their definitions. The following lines make a few adjustments as the mapping is not exact.
-- Some of the keywords (GPS, Interpolated) are used for confirming the match.
        SELECT * INTO tmp_rec3 FROM (SELECT x_y_position_code FROM pdb.public.position_code
          WHERE description = tmp_rec2.xy_code_desc) AS a3(xy_code);
        IF tmp_rec2.xy_code_desc IS NULL THEN
          tmp_rec3.xy_code = 'unknown';
        END IF;
        IF tmp_rec2.xy_code_desc ~ 'GPS' THEN
          tmp_rec3.xy_code = 'GPS';
        END IF;
        IF tmp_rec2.xy_code_desc ~ 'Interpolated' THEN
          tmp_rec3.xy_code = 'Interpolated';
        END IF;

-- We proceed to porting to PDB only for none NULL geometry
        IF tmp_rec.the_geom IS NOT NULL THEN

-- Creating a feature_id associated to this data package. Note that the data_package_id
-- is defined in this case as the cruise_id.
          tmp_featureid = createFeature('IP',getCruiseIDFromIDFilename(tmp_rec.id_filename));

-- Creation of an instantaneous_point entry.
          INSERT INTO pdb.public.instantaneous_point(feature_id, feature_code, cruise_id, point_type, date_time)
            VALUES (tmp_featureid, 'NADAS', getCruiseIDFromIDFilename(tmp_rec.id_filename), 1, tmp_rec.time);

-- The SRID is ported from its original 4269 value to 4326.
          tmp_geom = ST_Transform(tmp_rec.the_geom, 4326);

-- Creation of an measurement_location entry
          INSERT INTO pdb.public.measurement_location(feature_id, feature_code, geom, x_y_position_code)
            VALUES (tmp_featureid,'NADAS',tmp_geom, tmp_rec3.xy_code) RETURNING measurement_location_id
            INTO tmp_measurement_location_id;

-- Creation of PDB data (non null air_temperature_degc)
          IF tmp_rec.air_temperature_degc IS NOT NULL THEN
             INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
               group_id, data_value, quality_flag_id)
               VALUES (tmp_measurement_location_id,55,7,1,tmp_rec.air_temperature_degc,0);
          END IF;
```

```
-- Creation of PDB data (non null barometric_pressure_mbar)
          IF tmp_rec.barometric_pressure_mbar IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,52,7,1,tmp_rec.barometric_pressure_mbar,0);
          END IF;
-- Creation of PDB data (non null course_over_ground_degt)
          IF tmp_rec.course_over_ground_degt IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,135,7,1,tmp_rec.course_over_ground_degt,0);
          END IF;
-- Creation of PDB data (non null depth_m)
          IF tmp_rec.depth_m IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,308,7,1,tmp_rec.depth_m,0);
          END IF;
-- Creation of PDB data (non null percipitation_mm)
          IF tmp_rec.percipitation_mm IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,53,7,1,tmp_rec.percipitation_mm,0);
          END IF;
-- Creation of PDB data (non null relative_humidity_pct)
          IF tmp_rec.relative_humidity_pct IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,51,7,1,tmp_rec.relative_humidity_pct,0);
          END IF;
-- Creation of PDB data (non null ship_heading_degt)
          IF tmp_rec.ship_heading_degt IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,127,7,1,tmp_rec.ship_heading_degt,0);
          END IF;
-- Creation of PDB data (non null ship_propeller_port_rpm)
          IF tmp_rec.ship_propeller_port_rpm IS NOT NULL THEN
             INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
               group_id, data_value, quality_flag_id)
               VALUES (tmp_measurement_location_id,130,7,1,tmp_rec.ship_propeller_port_rpm,0);
          END IF;
-- Creation of PDB data (non null speed_over_ground_kt)
          IF tmp_rec.speed_over_ground_kt IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,134,7,1,tmp_rec.speed_over_ground_kt,0);
          END IF;
-- Creation of PDB data (non null surface_water_temperature_degc)
          IF tmp_rec.surface_water_temperature_degc IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,50,7,1,tmp_rec.surface_water_temperature_degc,0);
          END IF;
-- Creation of PDB data (non null wind_speed_kt)
          IF tmp_rec.wind_speed_kt IS NOT NULL THEN
             INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
               group_id, data_value, quality_flag_id)
               VALUES (tmp_measurement_location_id,128,7,1,tmp_rec.wind_speed_kt,0);
          END IF;
-- Creation of PDB data (non null wind_dir_degt)
          IF tmp_rec.wind_dir_degt IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
```

```
              VALUES (tmp_measurement_location_id,129,7,1,tmp_rec.wind_dir_degt,0);
          END IF;
-- Creation of PDB data (non null visibility_solar)
          IF tmp_rec.visibility_solar IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,57,7,1,tmp_rec.visibility_solar,0);
          END IF;
-- Creation of PDB data (non null ship_propeller_stbd_rpm)
          IF tmp_rec.ship_propeller_stbd_rpm IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,131,7,1,tmp_rec.ship_propeller_stbd_rpm,0);
          END IF;
        END IF;
      END IF;
    END LOOP;
    PERFORM dblink_close('nadas_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertNADAS(text,text,text) OWNER TO postgres;

-- Function that verifies that NADAS data are in the PDB
CREATE OR REPLACE FUNCTION verifyNADAS()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying NADAS data...';
-- Verification of the NADAS data value (parameter_id = 55 ) air_temperature_degc
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 55
      EXCEPT ALL SELECT b FROM  dblink('dbname=readbv3b user=postgres','SELECT air_temperature_degc
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL')  AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS air_temperature_degc data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 55';
    END IF;
-- Verification of the NADAS data value (parameter_id = 52 ) barometric_pressure_mbar
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 52
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT barometric_pressure_mbar
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS barometric_pressure_mbar data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 52';
    END IF;
-- Verification of the NADAS data value (parameter_id = 135 ) course_over_ground_degt
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 135
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT course_over_ground_degt
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS course_over_ground_degt data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 135';
    END IF;
-- Verification of the NADAS data value (parameter_id = 308 ) depth_m
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 308
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT depth_m
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
```

```
       RAISE NOTICE 'OK (NADAS depth_m data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 308';
    END IF;
-- Verification of the NADAS data value (parameter_id = 53 ) percipitation_mm
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 53
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT percipitation_mm
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS percipitation_mm data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 53';
    END IF;
-- Verification of the NADAS data value (parameter_id = 51 ) relative_humidity_pct
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 51
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT relative_humidity_pct
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL')  AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS relative_humidity_pct data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 51';
    END IF;
-- Verification of the NADAS data value (parameter_id = 127 ) ship_heading_degt
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 127
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT ship_heading_degt
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS ship_heading_degt data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 127';
    END IF;
-- Verification of the NADAS data value (parameter_id = 130 ) ship_propeller_port_rpm
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 130
      EXCEPT ALL SELECT b FROM  dblink('dbname=readbv3b user=postgres','SELECT ship_propeller_port_rpm
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS ship_propeller_port_rpm data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 130';
    END IF;
-- Verification of the NADAS data value (parameter_id = 134 ) speed_over_ground_kt
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 134
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT speed_over_ground_kt
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS speed_over_ground_kt data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 134';
    END IF;
-- Verification of the NADAS data value (parameter_id = 50 ) surface_water_temperature_degc
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 50
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT surface_water_temperature_degc
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS surface_water_temperature_degc data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 50';
    END IF;
-- Verification of the NADAS data value (parameter_id = 128 ) SELECT wind_speed_kt
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 128
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT wind_speed_kt
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL')  AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS wind_s--    This verification command was commented as it was not valid anymore due to additional
```

```
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.peed_kt data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 128';
    END IF;
-- Verification of the NADAS data value (parameter_id = 129 ) wind_dir_degt
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 129
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT wind_dir_degt
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS wind_dir_degt data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 129';
    END IF;
-- Verification of the NADAS data value (parameter_id = 57 ) visibility_solar
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 57
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT visibility_solar
        FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS visibility_solar data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 57';
    END IF;
-- Verification of the NADAS data value (parameter_id = 131 ) ship_propeller_stbd_rpm
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 131
      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT ship_propeller_stbd_rpm
      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (NADAS ship_propeller_stbd_rpm data)';
    ELSE
       RAISE NOTICE 'Missmatch found in NADAS data(data_value), parameter_id = 131';
    END IF;

--    This verification command was commented as it was not valid anymore due to additional
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.
--    SELECT count(*) INTO tmp_rec FROM (SELECT geom FROM measurement_location WHERE feature_code = 'NADAS'
--      EXCEPT ALL SELECT b FROM dblink('dbname=readbv3b user=postgres','SELECT the_geom
--      FROM readbv3b.public.nadas_observations WHERE the_geom IS NOT NULL') AS t(b geometry)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (NADAS geom)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in NADAS measurement_location(geom)';
--    END IF;
--    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyNADAS() OWNER TO postgres;

-- Function which deletes the data related to NADAS.
CREATE OR REPLACE FUNCTION deleteNADAS()
RETURNS VOID AS
 $BODY$
  BEGIN
    RAISE NOTICE 'Delete all data related to NADAS...';
    DELETE FROM data WHERE measuring_device_id = 7;
    RAISE NOTICE 'Delete measurement_location related to NADAS...';
    DELETE FROM measurement_location WHERE feature_code = 'NADAS';
    RAISE NOTICE 'Delete instantaneous_point related to NADAS...';
    DELETE FROM instantaneous_point WHERE feature_code = 'NADAS';
    RAISE NOTICE 'Done.';
  END;
$BODY$
```

```
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteNADAS() OWNER TO postgres;
```

This page intentionally left blank.

Annex N: ecs_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertECS()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_rec2 RECORD;
    tmp_rec3 RECORD;
    tmp_row_index integer;
    tmp_row_index2 integer;
    tmp_row_index3 integer;
    tmp_total_row_nb integer;
    tmp_total_row_nb2  integer;
    tmp_total_row_nb3  integer;
    tmp_featureid integer;
    tmp_surveyid integer;
    tmp_stationid integer;
    tmp_geom geometry;
    tmp_series_array integer[];
    tmp_series_index integer;
    tmp_measurement_location_id integer;
    tmp_bearing_array double precision[];
    tmp_range double precision;
    tmp_apex_coord decimal[];
    tmp_wind_estimate real[];
    tmp_wind_mean double precision[];
    tmp_wind_stdev double precision[];
    tmp_north_series_id integer;
    tmp_east_series_id integer;
    tmp_west_series_id integer;
    tmp_start_time timestamp without time zone;
    tmp_date_time timestamp without time zone;
    tmp_time_left varchar(200);
    tmp_data_package_id integer;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');

--  Opening a connection to the LDB ecs_flights with a cursor at the beginning of the table
    PERFORM dblink_open('ecs_flights_cursor', 'SELECT id, date, windsite1, windsite2, windsite3,
      windsite4 FROM readbv3b.public.ecs_flights');
--  Creation of the data package ID and insertion of a new entry in the data_package table
    tmp_data_package_id = 2000001;
    INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
      VALUES (tmp_data_package_id,'ECS',
     'Data package associated to the Eastern Canada shallow (ECS) water ambient noise dataset');
```

```
-- Creation of three entries in the series table, one for each apex of the equilateral triangle.
-- The triangle is point to the north, the other apexes point to the west and east.
-- The series_id is kept to feed an array to be reused latter on in this script.
    INSERT INTO pdb.public.series(description) VALUES ('Northern point on triangle')
      RETURNING series_id INTO tmp_north_series_id;
    INSERT INTO pdb.public.series(description) VALUES ('Eastern point on triangle')
      RETURNING series_id INTO tmp_east_series_id;
    INSERT INTO pdb.public.series(description) VALUES ('Western point on triangle')
      RETURNING series_id INTO tmp_west_series_id;
    tmp_series_array = ARRAY[tmp_north_series_id,tmp_east_series_id,tmp_west_series_id];
    tmp_bearing_array = ARRAY[0.0,120.0,240.0];
--  Distance in nautical miles between one of the apexes of an equilateral triangle (with 10 NM sides)
--  and its center
    tmp_range = 10.0/sqrt(3);

-- This command retrieves the total number of flights, this number is used in the following loop.
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*) FROM
      readbv3b.public.ecs_flights ') AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;

-- Loop over all (12) flights
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

-- Standard script for measuring and printing the time for executing this script
      tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
      RAISE NOTICE 'ECS: %', tmp_time_left;

-- Fetching one row of the ecs_flights table. The wind estimates is used to fill an array
-- of dimension 4, the number of sites (defined here as station)
      SELECT * INTO tmp_rec FROM dblink_fetch('ecs_flights_cursor',1)
        AS a(id integer, date timestamp without time zone, windsite1 real, windsite2 real, windsite3 real, windsite4 real);
      tmp_wind_estimate = ARRAY[tmp_rec.windsite1,tmp_rec.windsite2,tmp_rec.windsite3,tmp_rec.windsite4];

-- We remove the hours (23h) to round it up to 00h, in the process we add a full day to the date.
      tmp_date_time = date_trunc('day', tmp_rec.date) + interval '1 day';

-- For each flight, we create a PDB survey entry and we return the survey_id for further use in this script.
-- We fill the start and end time of the survey from the information found in the ecs_flights table
      INSERT INTO pdb.public.survey(start_date_time, end_date_time,description, measuring_device_id)
        VALUES (tmp_date_time,tmp_date_time,'Eastern Canada ambient noise Flight '||tmp_rec.id,6)
        RETURNING survey_id INTO tmp_surveyid;

-- This command retrieves the total number of sites (4), this number is used in the following loop.
      SELECT * INTO tmp_rec2 FROM dblink('dbname=readbv3b user=postgres','SELECT count(*) FROM
        readbv3b.public.ecs_sites ') AS a(b integer);
      tmp_total_row_nb2 = tmp_rec2.b;

--  Setting up a cursor in the ecs_sites LDB table in order to loop over its content
      PERFORM dblink_open('ecs_sites_cursor', 'SELECT id,the_geom,location,sediment,depth_m,shipping
        FROM readbv3b.public.ecs_sites');

-- Loop over all (4) sites (defined as stations in the PDB)
      FOR tmp_row_index2 IN 1..tmp_total_row_nb2 LOOP

-- Fetching all the information in the ecs_sites table.
        SELECT * INTO tmp_rec2 FROM dblink_fetch('ecs_sites_cursor',1)
          AS a(id integer,the_geom geometry,location text,
          sediment text, depth_m real, shipping text);

-- Associating the station ID as the ecs_sites ID
        tmp_stationid = tmp_rec2.id;

-- Loop over the 3 apexes of the triangle (1=north, 2=east, 3=west)
```

```
        FOR tmp_series_index IN 1..3 LOOP

-- Creation of the feature ID for that location as in Instantaneous Point (IP).
          tmp_featureid = createFeature('IP',tmp_data_package_id);

-- Creation of an instantaneous point entry for each apex(series), for each flight(survey) and site(station)
          INSERT INTO pdb.public.instantaneous_point(feature_id, feature_code, station_id, survey_id,
            series_id, point_type, date_time)   VALUES ( tmp_featureid, 'AMBIENTNOISE', tmp_stationid,
            tmp_surveyid, tmp_series_array[tmp_series_index], 1, tmp_date_time);

-- Creation of an ambient_noise entry for each apex(series), for each flight(survey) and site(station)
-- There is here a volontary duplication of the data
          INSERT INTO pdb.public.ambient_noise(feature_id, location_comment, sediment_comment,
            approximate_depth, shipping_comment)
            VALUES ( tmp_featureid , tmp_rec2.location, tmp_rec2.sediment, tmp_rec2.depth_m, tmp_rec2.shipping );

-- Computation of the apex coordinates from the site(station) geom
          tmp_apex_coord = getCoordFromPointRangeBearing(ST_Y(tmp_rec2.the_geom), ST_X(tmp_rec2.the_geom),
            tmp_bearing_array[tmp_series_index], tmp_range);

-- Creation of a new 3D geom associated to the apex of the triangle associated to a depth of 30 meter.
          tmp_geom = GeomFromEWKT('SRID=4326;POINT('||tmp_apex_coord[2]||' '|| tmp_apex_coord[1] ||' '||'30)');

-- Creation of a measurement location entry for each apex(series), for each flight(survey) and site(station)
-- using the new calculated apex geom. The returned measurement_location_id is latter used for storing the data.
          INSERT INTO pdb.public.measurement_location(feature_id, feature_code, geom, x_y_position_code)
            VALUES (tmp_featureid,'AMBIENTNOISE',tmp_geom, 'computed') RETURNING measurement_location_id
            INTO tmp_measurement_location_id;

          IF tmp_wind_estimate[tmp_stationid] IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,800,6,1,tmp_wind_estimate[tmp_stationid],0);
          END IF;

-- Setting up a cursor in the ecs_wind_obs LDB table in order to loop over its content.
-- Only the data associated to a specific flight (from the first loop) and site
-- (from the second loop) is selected.
        PERFORM dblink_open('ecs_wind_obs_cursor', 'SELECT id_flight, id_site, hz, north_mean, north_stdev,
          east_mean, east_stdev, west_mean, west_stdev FROM readbv3b.public.ecs_wind_obs
          WHERE id_flight='|| tmp_rec.id||' AND id_site = '||tmp_stationid);

-- This command retrieves the total number of ecs_wind_obs for given flight
          SELECT * INTO tmp_rec3 FROM dblink('dbname=readbv3b user=postgres',
            'SELECT count(*) FROM readbv3b.public.ecs_wind_obs WHERE id_flight='||
            tmp_rec.id ||' AND id_site = '||tmp_stationid) AS a(b integer);

          tmp_total_row_nb3 = tmp_rec3.b;

-- Looping over all the ecs_wind_obs rows associated to a given flight
          FOR tmp_row_index3 IN 1..tmp_total_row_nb3 LOOP
-- Fetching one row of the ecs_wind_obs table.
            SELECT * INTO tmp_rec3 FROM dblink_fetch('ecs_wind_obs_cursor',1)
              AS (id_flight integer, id_site integer, hz double precision, north_mean double precision,
              north_stdev double precision, east_mean double precision, east_stdev double precision,
              west_mean double precision, west_stdev double precision);
-- Aligning the mean and stdev in arrays, to fit with the triangle apexes.
            tmp_wind_mean = ARRAY[tmp_rec3.north_mean,tmp_rec3.east_mean,tmp_rec3.west_mean];
            tmp_wind_stdev = ARRAY[tmp_rec3.north_stdev, tmp_rec3.east_stdev, tmp_rec3.west_stdev];
-- Creating a data PDB entry. We skip null value and frequency 0
            IF tmp_rec3.hz IS NOT NULL AND tmp_rec3.hz <> 0 THEN
              INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
                group_id, data_value, quality_flag_id)
                VALUES (tmp_measurement_location_id,810,6,tmp_row_index3,tmp_rec3.hz,0);
```

```
            END IF;
-- Creating a data PDB entry. We skip null value
            IF tmp_wind_mean[tmp_series_index] IS NOT NULL THEN
              INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
                group_id, data_value, quality_flag_id)
                VALUES (tmp_measurement_location_id,801,6,tmp_row_index3,tmp_wind_mean[tmp_series_index],0);
            END IF;
-- Creating a data PDB entry. We skip null value
            IF tmp_wind_stdev[tmp_series_index] IS NOT NULL THEN
              INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
                group_id, data_value, quality_flag_id)
                VALUES (tmp_measurement_location_id,802,6,tmp_row_index3,tmp_wind_stdev[tmp_series_index],0);
            END IF;

          END LOOP;
          PERFORM dblink_close('ecs_wind_obs_cursor');
        END LOOP;
      END LOOP;
    PERFORM dblink_close('ecs_sites_cursor');
    END LOOP;
    PERFORM dblink_close('ecs_flights_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertECS() OWNER TO postgres;

-- Function that verifies that ECS data are in the PDB
CREATE OR REPLACE FUNCTION verifyECS()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying ECS (ambient_noise) data...';
--    This verification command was commented as it was not valid anymore due to additional
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.
--    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 810 EXCEPT ALL SELECT b FROM
--      dblink('dbname=readbv3b user=postgres','SELECT hz FROM readbv3b.public.ecs_wind_obs') AS t(b double precision)) AS t2(b2)
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (ECS hz data)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in ECS data(data_value), parameter_id = 810';
--    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifyECS() OWNER TO postgres;

-- Function which deletes the ambient_noise table and related data.
CREATE OR REPLACE FUNCTION deleteECS()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting measuring_device related to ECS(ambient_noise)...';
    DELETE FROM data WHERE measuring_device_id = 6;
    RAISE NOTICE 'Deleting measurement_location related to ECS(ambient_noise)...';
    DELETE FROM measurement_location WHERE feature_code = 'AMBIENTNOISE';
    RAISE NOTICE 'Deleting ambient_noise...';
    DELETE FROM ambient_noise;
    RAISE NOTICE 'Deleting instantaneous_point related to ECS(ambient_noise)...';
    DELETE FROM instantaneous_point WHERE feature_code = 'AMBIENTNOISE';
```

```
    RAISE NOTICE 'Deleting survey related to ECS(ambient_noise)...';
    DELETE FROM survey WHERE measuring_device_id = 6;
    RAISE NOTICE 'Deleting series related to ECS(ambient_noise)...';
    DELETE FROM series WHERE description = 'Northern point on triangle';
    DELETE FROM series WHERE description = 'Eastern point on triangle';
    DELETE FROM series WHERE description = 'Western point on triangle';
    DELETE FROM feature WHERE data_package_id = 2000001;
    RAISE NOTICE 'Deleting data_package_id related to ECS(ambient_noise)...';
    DELETE FROM data_package WHERE data_package_id = 2000001;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteECS() OWNER TO postgres;
```

This page intentionally left blank.

Annex O: swdb_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

-- getSWDBArrayValue
-- Function that read a string column_name from a file identified by filenameid
-- which returns the nth element (identified by column_number) of an ARRAY
-- from casting the input string
CREATE OR REPLACE FUNCTION getSWDBArrayValue(column_name char(100), filenameid integer, column_number integer)
RETURNS  decimal AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    rec RECORD;
    out_dec decimal[];
  BEGIN
-- Selection the array string from the file
    SELECT * INTO rec FROM dblink('dbname=readbv3b user=postgres','SELECT '|| column_name ||
      ' FROM readbv3b.public.swdb_index  WHERE id_filename = '|| filenameid) AS a(out text);
-- Castting the strings into an array
    out_dec = string_to_array(rec.out,',');
-- Returning the element identified by column_number
    RETURN out_dec[column_number];
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getSWDBArrayValue(char(100), integer, integer) OWNER TO postgres;

-- getSWDBArrayLength
-- Function that return the number of hydrophones from a file identified by filenameid
-- which correspond to the array dimension
CREATE OR REPLACE FUNCTION getSWDBArrayLength(filenameid integer)
RETURNS  integer AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    rec RECORD;
  BEGIN
-- Selection the number of hydrophones from the table swdb_index for the file identified by filenameid
    SELECT * INTO rec FROM dblink('dbname=readbv3b user=postgres',
      'SELECT nhydrophones FROM readbv3b.public.swdb_index WHERE id_filename = '|| filenameid)
      AS a(array_length integer);
-- Returning the length of the array (number of hydrophones)
    RETURN rec.array_length;
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION getSWDBArrayLength(integer) OWNER TO postgres;

-- insertSWDBArray

CREATE OR REPLACE FUNCTION insertSWDB()
RETURNS VOID AS
```

```
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_rec2 RECORD;
    tmp_row_index integer;
    tmp_row_index2 integer;
    tmp_freq_index integer;
    tmp_total_row_nb integer;
    tmp_total_row_nb2  integer;
    tmp_featureid integer;
    tmp_stationid integer;
    tmp_array_index integer;
    tmp_measuring_device_id bigint;
    tmp_hp_nb decimal;
    tmp_hp_depth decimal;
    tmp_hp_pos decimal;
    tmp_idfilename integer;
    tmp_geom_xyz geometry;
    tmp_measurement_location_id integer;
    tmp_start_date_time timestamp without time zone;
    tmp_end_date_time timestamp without time zone;
    tmp_edit_date_time timestamp without time zone;
    tmp_bearing decimal;
    tmp_source_depth decimal;
    tmp_station_depth decimal;
    tmp_minimum_run_depth decimal;
    tmp_maximum_run_depth decimal;
    tmp_bottom_type varchar(200);
    tmp_sound_description varchar(200);
    tmp_sea_state integer;
    tmp_line_direction varchar(20);
    tmp_minimum_range decimal;
    tmp_maximum_range decimal;
    tmp_creation_date_time timestamp without time zone;
    tmp_no_of_shots integer;
    tmp_no_of_frequencies integer;
    tmp_bandwidth varchar(20);
    tmp_survey_id integer;
    tmp_range_data real;
    tmp_frequency_data decimal[];
    tmp_transloss_data decimal[];
    tmp_description varchar(2000);
    tmp_long decimal;
    tmp_lat decimal;
    tmp_str varchar(100);
    tmp_start_time timestamp without time zone;
    tmp_time_left varchar(200);
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB swdb_index with a cursor at the beginning of the table
    PERFORM dblink_open('swdb_index_cursor', 'SELECT id_filename, tag_station_number, tag_area_name,
      tag_latitude, tag_longitude, tag_run_number, tag_st_date, tag_st_time, tag_en_time, tag_bearing,
      tag_source_depth,tag_sta_depth, tag_min_depth, tag_max_depth, tag_bottom_type, tag_ss_profile,
      tag_sea_state, tag_direction, tag_min_range, tag_max_range, tag_bandwidth, tag_creation_date,
      tag_creation_time, tag_date_last_edit, tag_time_last_edit, tag_no_of_shots, tag_no_of_frequencies,
      tag_frequency, tag_run_number_extension, tag_notes, id_cruise, tag_ranges, tag_shot_numbers
      FROM readbv3b.public.swdb_index');

-- Counting the total number of rows of swdb_index, needed for looping on the table
```

```
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*)
      FROM readbv3b.public.swdb_index ') AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;

-- Looping on all swdb_index entries
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

-- Initialization of the temporary variables needed within the loop
      tmp_featureid = NULL;
      tmp_stationid = NULL;
      tmp_idfilename = NULL;
      tmp_bearing = NULL;
      tmp_source_depth = NULL;
      tmp_station_depth = NULL;
      tmp_minimum_run_depth = NULL;
      tmp_maximum_run_depth = NULL;
      tmp_bottom_type = NULL;
      tmp_sound_description = NULL;
      tmp_sea_state = NULL;
      tmp_line_direction = NULL;
      tmp_minimum_range = NULL;
      tmp_maximum_range = NULL;
      tmp_creation_date_time = NULL;
      tmp_edit_date_time = NULL;
      tmp_no_of_shots = NULL;
      tmp_no_of_frequencies = NULL;
      tmp_bandwidth = NULL;
      tmp_description = NULL;

-- Standard script for measuring and printing the time for executing this script
      tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
      RAISE NOTICE 'SWDB: %', tmp_time_left;

-- Fetching one row of the swdb_index table and storing the result in tmp_rec
      SELECT * INTO tmp_rec FROM dblink_fetch('swdb_index_cursor',1) AS a1(id_filename integer,
        tag_station_number text, tag_area_name text, tag_latitude text, tag_longitude text, tag_run_number text,
        tag_st_date text, tag_st_time text, tag_en_time text, tag_bearing text,tag_source_depth text,
        tag_sta_depth text, tag_min_depth text, tag_max_depth text, tag_bottom_type text, tag_ss_profile text,
        tag_sea_state text, tag_direction text, tag_min_range text, tag_max_range text, tag_bandwidth text,
        tag_creation_date text, tag_creation_time text, tag_date_last_edit text, tag_time_last_edit text,
        tag_no_of_shots text, tag_no_of_frequencies text, tag_frequency text, tag_run_number_extension text,
        tag_notes text, id_cruise integer, tag_ranges text, tag_shot_numbers text);

-- Setting the tmp_idfilename
      tmp_idfilename = tmp_rec.id_filename;

-- Casting the string array of frequencies into a SQL array
      tmp_frequency_data = string_to_array(tmp_rec.tag_frequency,',');

-- Looping on all the hydrophones
      FOR tmp_array_index IN 1..getSWDBArrayLength(tmp_idfilename) LOOP
-- Initializing the tmp_str variable
        tmp_str = NULL;

-- For a given idfilename, we retrieve the parameters for each hydrophone of the layout,
-- its idenfication number, its depth and its horizontal position if any.
        tmp_hp_nb = getSWDBArrayValue('tag_hp_numbers', tmp_idfilename, tmp_array_index);
        tmp_hp_depth = getSWDBArrayValue('tag_hp_depth', tmp_idfilename, tmp_array_index);
        tmp_hp_pos = getSWDBArrayValue('tag_hp_position', tmp_idfilename, tmp_array_index);

-- If no horizontal is found, we assumed that there is no horizontal hydrophones set up.
        IF tmp_hp_pos IS NULL THEN
          tmp_hp_pos = 0;
        END IF;
```

```
-- We create a unique identification string for each of the hydrophones base on the 3 above parameters:
-- its idenfication number, its depth and its horizontal position. The identification number always
-- start we a 1 followed by 2 digits for the ID number, 3 digits for the depth and 3 digits for the horizontal
-- position.
       tmp_str = '1'||trim(to_char(tmp_hp_nb,'09'))||trim(to_char(tmp_hp_depth,'099V9'))||
       trim(to_char(tmp_hp_pos,'099V9'));
-- The identification string is cast into a large number (bigint) and will become the measuring_device_id.
       tmp_measuring_device_id = CAST( tmp_str AS bigint);
-- If the measuring_device_id identify by the long ID strings is not found in the measuring device table,
-- it is then created, as well as the 3 parameters that are stored as device_class_detail.
       SELECT * INTO tmp_rec2 FROM measuring_device WHERE measuring_device_id = tmp_measuring_device_id;
       IF NOT FOUND THEN
         INSERT INTO pdb.public.measuring_device(measuring_device_id, name, description) VALUES
           (tmp_measuring_device_id, 'HYDROPHONE', 'Hydrophone with hp_number = '||tmp_hp_nb||' hp_depth = '||
           tmp_hp_depth||' and hp_position = '||tmp_hp_pos);
         INSERT INTO pdb.public.device_class_detail(measuring_device_id, descriptor, value)
           VALUES (tmp_measuring_device_id ,'HPNumber', tmp_hp_nb);
         INSERT INTO pdb.public.device_class_detail(measuring_device_id, descriptor, value)
           VALUES (tmp_measuring_device_id ,'HPDepth', tmp_hp_depth );
         INSERT INTO pdb.public.device_class_detail(measuring_device_id, descriptor, value)
           VALUES (tmp_measuring_device_id ,'HPPosition', tmp_hp_pos);
       END IF;
     END LOOP;
-- If the tag_station_number string contains numerals, then it is cast into a number
     IF  tmp_rec.tag_station_number ~ '[0-9]' THEN
       tmp_stationid := to_number(tmp_rec.tag_station_number,'9999999');
     END IF;
-- If the tag_st_date and tag_st_time contains numerals, then it is cast together into a timestamp
     IF  tmp_rec.tag_st_date ~ '[0-9]' AND tmp_rec.tag_st_time ~ '[0-9]' THEN
       tmp_start_date_time := createTimestamp(tmp_rec.tag_st_date,tmp_rec.tag_st_time);
     END IF;
-- If the tag_st_date and tag_en_time contains numerals, then it is cast together into a timestamp
     IF  tmp_rec.tag_st_date ~ '[0-9]' AND tmp_rec.tag_en_time ~ '[0-9]' THEN
       tmp_end_date_time := createTimestamp(tmp_rec.tag_st_date,tmp_rec.tag_en_time);
     -- In order to create a proper end timestamp, the start date is used to generate the
     -- timestamp since there is no end date provided in the LDB. If the end time has crossed
     -- the Change Date Line, an additional 24 hours is added to the end timestamp
       IF tmp_end_date_time < tmp_start_date_time  THEN
         tmp_end_date_time := tmp_end_date_time + interval '24 hours';
       END IF;
     END IF;

-- If the tag_bearing string contains numerals, then it is cast into a number
     IF  tmp_rec.tag_bearing ~ '[0-9]' THEN
       tmp_bearing = to_number(tmp_rec.tag_bearing,'9999.99');
     END IF;
-- If the tag_source_depth string contains numerals, then it is cast into a number
     IF  tmp_rec.tag_source_depth ~ '[0-9]' THEN
       tmp_source_depth = to_number(tmp_rec.tag_source_depth,'9999.99');
     END IF;
-- If the tag_sta_depth string contains numerals, then it is cast into a number
     IF  tmp_rec.tag_sta_depth ~ '[0-9]' THEN
       tmp_station_depth = to_number(tmp_rec.tag_sta_depth,'9999.99');
     END IF;
-- If the tag_min_depth string contains numerals, then it is cast into a number
     IF  tmp_rec.tag_min_depth ~ '[0-9]' THEN
       tmp_minimum_run_depth = to_number(tmp_rec.tag_min_depth,'9999.99');
     END IF;
-- If the tag_max_depth string contains numerals, then it is cast into a number
     IF  tmp_rec.tag_max_depth ~ '[0-9]' THEN
       tmp_maximum_run_depth = to_number(tmp_rec.tag_max_depth,'9999.99');
     END IF;
-- If the tag_bottom_type string is not null, it is copied to tmp_bottom_type
```

```
      IF  tmp_rec.tag_bottom_type IS NOT NULL THEN
        tmp_bottom_type = tmp_rec.tag_bottom_type;
      END IF;
-- If the tag_ss_profile string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_ss_profile IS NOT NULL THEN
        tmp_sound_description = tmp_rec.tag_ss_profile;
      END IF;
-- If the tag_sea_state string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_sea_state ~ '[0-9]' THEN
        tmp_sea_state = to_number(tmp_rec.tag_sea_state,'99');
      END IF;
-- If the tag_direction string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_direction IS NOT NULL THEN
        tmp_line_direction = tmp_rec.tag_direction;
      END IF;
-- If the tag_min_range string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_min_range ~ '[0-9]' THEN
        tmp_minimum_range = to_number(tmp_rec.tag_min_range,'99999.99');
      END IF;
-- If the tag_max_range string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_max_range ~ '[0-9]' THEN
        tmp_maximum_range = to_number(tmp_rec.tag_max_range,'99999.99');
      END IF;
-- If the tag_creation_date string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_creation_date ~ '[0-9]' AND tmp_rec.tag_creation_time ~ '[0-9]' THEN
        tmp_creation_date_time := createTimestamp(tmp_rec.tag_creation_date,tmp_rec.tag_creation_time);
      END IF;
-- If the tag_date_last_edit string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_date_last_edit ~ '[0-9]' AND tmp_rec.tag_time_last_edit ~ '[0-9]' THEN
        tmp_edit_date_time := createTimestamp(tmp_rec.tag_date_last_edit,tmp_rec.tag_time_last_edit);
      END IF;
-- If the tag_no_of_shots string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_no_of_shots ~ '[0-9]' THEN
        tmp_no_of_shots = to_number(tmp_rec.tag_no_of_shots,'999');
      END IF;
-- If the tag_no_of_frequencies string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_no_of_frequencies ~ '[0-9]' THEN
        tmp_no_of_frequencies = to_number(tmp_rec.tag_no_of_frequencies,'999');
      END IF;
-- If the tag_bandwidth string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_bandwidth IS NOT NULL THEN
        tmp_bandwidth = tmp_rec.tag_bandwidth;
      END IF;
-- If the tag_area_name string contains numerals, then it is cast into a number
      IF  tmp_rec.tag_area_name IS NOT NULL OR tmp_rec.tag_run_number IS NOT NULL THEN
        tmp_description = 'Area = ' || tmp_rec.tag_area_name || ' with run number = ' || tmp_rec.tag_run_number
        || tmp_rec.tag_run_number_extension;
      END IF;

--  Creating a feature_id associated to this data package. Note that the data_package_id
--  is defined in this case as the cruise_id
      tmp_featureid = createFeature('IP',tmp_rec.id_cruise);

-- Creation of an instantaneous_point entry.
      INSERT INTO pdb.public.instantaneous_point(feature_id,feature_code,cruise_id,station_id,point_type,date_time)
        VALUES ( tmp_featureid, 'TRANLOSS', tmp_rec.id_cruise, tmp_stationid, 1, tmp_start_date_time);

-- Creation of a new row in the transmission_loss table
      INSERT INTO pdb.public.transmission_loss(feature_id, bearing,source_depth, station_depth, minimum_run_depth,
        maximum_run_depth, bottom_type, sound_description, sea_state, line_direction, minimum_range, maximum_range,
        creation_date_time, edit_date_time,no_of_shots, no_of_frequencies, bandwidth, original_station_id, start_date_time, end_d
        VALUES (tmp_featureid, tmp_bearing, tmp_source_depth, tmp_station_depth, tmp_minimum_run_depth,
        tmp_maximum_run_depth, tmp_bottom_type, tmp_sound_description, tmp_sea_state, tmp_line_direction,
        tmp_minimum_range, tmp_maximum_range, tmp_creation_date_time, tmp_edit_date_time, tmp_no_of_shots,
```

```
        tmp_no_of_frequencies, tmp_bandwidth, tmp_rec.tag_station_number,tmp_start_date_time,tmp_end_date_time);

-- Creation of a new row in the survey table and storing the date entries
      INSERT INTO pdb.public.survey( description, start_date_time, end_date_time)
        VALUES (tmp_description, tmp_start_date_time, tmp_end_date_time) RETURNING survey_id  INTO tmp_survey_id;

-- The creation of the survey_key as been postponed
--      INSERT INTO pdb.public.survey_key( feature_id, survey_id) VALUES (tmp_featureid, tmp_survey_id);

-- Counting the total number of rows of swdb_tl_files for a given idfilename (needed for the following loop)
      SELECT * INTO tmp_rec2 FROM dblink('dbname=readbv3b user=postgres',
        'SELECT count(*) FROM readbv3b.public.swdb_tl_files WHERE id_filename ='|| tmp_idfilename ) AS a(b integer);
      tmp_total_row_nb2 = tmp_rec2.b;

-- Looping on all the lines associated to a given idfilename in the swdb_tl_files table
      FOR tmp_row_index2 IN 4..tmp_total_row_nb2 LOOP

-- Initialization of temporary variables needed in this sub-loop
        tmp_long = NULL;
        tmp_lat = NULL;

-- Opening a connection to the LDB swdb_tl_files with a cursor at the beginning of the rows
-- associated to a given idfilename and a given fileline (tmp_row_index2)
        PERFORM dblink_open('swdb_files_cursor', 'SELECT fileline,filecontent,gid_geopoint,id_shot,
          id_hydrophone,range FROM readbv3b.public.swdb_tl_files WHERE id_filename = '||tmp_idfilename||
          ' AND fileline = '||tmp_row_index2);

-- Fetching the given fileline of the idfilename in the swdb_tl_files table
        SELECT * INTO tmp_rec2 FROM  dblink_fetch('swdb_files_cursor',1) AS a2(fileline integer,
          filecontent varchar(200), gid_geopoint integer, id_shot integer, id_hydrophone integer, range real);

-- The strings for longitude and latitude are casted into decimal numbers.
tmp_long = to_number(tmp_rec.tag_longitude,'S999.9999999');
tmp_lat = to_number(tmp_rec.tag_latitude,'S99.9999999');

-- A 3D geometry is created based on the 2D geometry and the source depth. With SRID = 4326.
        tmp_geom_xyz = GeomFromEWKT('SRID=4326;POINT('||tmp_long||' '|| tmp_lat||' '|| tmp_source_depth  ||')');

-- The hydrophone long ID number is regenerated again. The first time was to create a measuring_device entry.
-- See that above section for further details. Now the same ID is generated to be used for storing the data.
        tmp_str = NULL;
        tmp_array_index = tmp_rec2.id_hydrophone;
        tmp_hp_nb = getSWDBArrayValue('tag_hp_numbers', tmp_idfilename, tmp_array_index);
        tmp_hp_depth = getSWDBArrayValue('tag_hp_depth', tmp_idfilename, tmp_array_index);
        tmp_hp_pos = getSWDBArrayValue('tag_hp_position', tmp_idfilename, tmp_array_index);
        IF tmp_hp_pos IS NULL THEN
          tmp_hp_pos = 0;
        END IF;
        tmp_str = '1'||trim(to_char(tmp_hp_nb,'09'))||trim(to_char(tmp_hp_depth,'099V9'))||
          trim(to_char(tmp_hp_pos,'099V9'));
        tmp_measuring_device_id = CAST( tmp_str AS bigint);
        tmp_range_data = tmp_rec2.range;

-- The transmission_loss data string and separated by commas is casted into an SQL array.
        IF tmp_rec2.filecontent ~ ',' THEN
          tmp_transloss_data = string_to_array(trim(tmp_rec2.filecontent),',');
        ELSE
-- The transmission_loss data string and separated by spaces is casted into an SQL array.
-- The given formula is based on the assumption of equal spacing
          FOR tmp_freq_index IN 1..tmp_no_of_frequencies LOOP
            tmp_transloss_data[tmp_freq_index] =
              to_number(substring(tmp_rec2.filecontent FROM (1+(tmp_freq_index-1)*5) for 5),'999.9');
          END LOOP;
        END IF;
```

```
-- Creation of an measurement_location entry using the new 3D geometry
        INSERT INTO pdb.public.measurement_location(feature_id, feature_code, geom, x_y_position_code)
          VALUES (tmp_featureid,'TRANLOSS',tmp_geom_xyz, 'computed') RETURNING measurement_location_id
          INTO tmp_measurement_location_id;

-- Looping on all the frequency array
        FOR tmp_freq_index IN 1..tmp_no_of_frequencies LOOP
-- Creation of a new PDB data entry: range (811)
          INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
            group_id, data_value, quality_flag_id)
            VALUES (tmp_measurement_location_id,811,tmp_measuring_device_id,tmp_freq_index,tmp_range_data,0);
-- Creation of a new PDB data entry: bearing (813)
          IF tmp_bearing IS NOT NULL THEN
            INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
              group_id, data_value, quality_flag_id)
              VALUES (tmp_measurement_location_id,813,tmp_measuring_device_id,tmp_freq_index,tmp_bearing,0);
          END IF;
-- Creation of a new PDB data entry: frequency (815)
          INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
            group_id, data_value, quality_flag_id)
            VALUES (tmp_measurement_location_id,815,tmp_measuring_device_id,tmp_freq_index,
            tmp_frequency_data[tmp_freq_index],0);
-- Creation of a new PDB data entry: transmission_loss (814)
          INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
            group_id, data_value, quality_flag_id)
            VALUES (tmp_measurement_location_id,814,tmp_measuring_device_id,tmp_freq_index,
            tmp_transloss_data[tmp_freq_index],0);
        END LOOP;
        PERFORM dblink_close('swdb_files_cursor');
      END LOOP;
    END LOOP;
    PERFORM dblink_close('swdb_index_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertSWDB() OWNER TO postgres;

-- Function that verifies that SWDB data are in the PDB
CREATE OR REPLACE FUNCTION verifySWDB()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Transmission Loss (SWDB) data ...';
--    This verification command was commented as it was not valid anymore due to additional
--    preprocessing changes added recently. New line of codes is needed for making the verification possible
--    in order to compensate for the preprocessing added.
--    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 811
--      EXCEPT ALL SELECT b FROM  dblink('dbname=readbv3b user=postgres','SELECT range
--      FROM readbv3b.public.swdb_tl_files') AS t(b real)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (SWDB range data)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in SWDB data(data_value), parameter_id = 811';
--    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifySWDB() OWNER TO postgres;
```

```
-- Function which deletes the data related to Transmission_Loss and SWDB.
CREATE OR REPLACE FUNCTION deleteSWDB()
RETURNS VOID AS
$BODY$
  BEGIN
    RAISE NOTICE 'Deleting data related to Transmission Loss (range, bearing, frequency, transmission loss)...';
    DELETE FROM data WHERE measuring_device_id > 1000000;
    RAISE NOTICE 'Deleting measurement_location related to Transmission Loss...';
    DELETE FROM measurement_location WHERE feature_code = 'TRANLOSS';
    RAISE NOTICE 'Deleting Transmission Loss metadata...';
    DELETE FROM transmission_loss;
    RAISE NOTICE 'Deleting survey related to Transmission Loss...';
    DELETE FROM survey WHERE description ~ 'with run number';
    RAISE NOTICE 'Deleting instantaneous_point related to Transmission Loss...';
    DELETE FROM instantaneous_point WHERE feature_code = 'TRANLOSS';
    RAISE NOTICE 'Deleting device_class_detail related to Transmission Loss (hydrophones)...';
    DELETE FROM device_class_detail WHERE measuring_device_id > 1000000;
    RAISE NOTICE 'Deleting measuring_device related to Transmission Loss (hydrophones)...';
    DELETE FROM measuring_device WHERE measuring_device_id > 1000000;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteSWDB() OWNER TO postgres;
```

Annex P: sediment_ss_fct.sql

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

CREATE OR REPLACE FUNCTION insertSedimentSS()
RETURNS VOID AS
$BODY$
  DECLARE
--  Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
    tmp_row_index integer;
    tmp_total_row_nb  integer;
    tmp_featureid integer;
    tmp_measurement_location_id integer;
    tmp_date_time timestamp without time zone;
    tmp_start_time timestamp without time zone;
    tmp_time_left varchar(200);
    tmp_data_package_id integer;
    tmp_array integer[];
    tmp_geom geometry;
    tmp_already_included_flag integer;
  BEGIN
--  Initialization of the clock for monitoring the porting process
    tmp_start_time = clock_timestamp();

--  Creation of the data package ID and insertion of a new entry in the data_package table
    tmp_data_package_id = 5000001;
    INSERT INTO pdb.public.data_package(data_package_id,data_package_name,description)
      VALUES (tmp_data_package_id,'SedimentSS','Data package associated to Scotia Shelf Sediment dataset');

--  Opening a connection to the READBv3b database as user postgres
    PERFORM dblink_connect('dbname=readbv3b user=postgres');
--  Opening a connection to the LDB sediment_ss_position with a cursor at the beginning of the table
    PERFORM dblink_open('sediment_ss_cursor', 'SELECT group_num, num_measurements, velocity_kps, error_kps,
        high_cluster, low_cluster, average_clearance_m, the_geom FROM readbv3b.public.sediment_ss_position');

-- Counting the total number of rows of sediment_ss_position, needed for looping on the table
    SELECT * INTO tmp_rec FROM dblink('dbname=readbv3b user=postgres','SELECT count(*)
      FROM readbv3b.public.sediment_ss_position ') AS a(b integer);
    tmp_total_row_nb = tmp_rec.b;
-- Creating a fix timestamp for the porting. Date was chosen after discussion.
    tmp_date_time = to_timestamp('860701000000','YYMMDDHHMISS');

-- Flag to prevent the inclusion of the second of two identical rows found in the LDB
    tmp_already_included_flag = 0;

-- Looping on all sediment_ss_position entries
    FOR tmp_row_index IN 1..tmp_total_row_nb LOOP

-- Standard script for measuring and printing the time for executing this script
      tmp_time_left = printTimeLeft(tmp_row_index, tmp_total_row_nb,tmp_start_time);
      RAISE NOTICE 'Sediment SS: %', tmp_time_left;
```

```
-- Fetching one row of the sediment_ss_position table and storing the result in tmp_rec
      SELECT * INTO tmp_rec FROM dblink_fetch('sediment_ss_cursor',1) AS a1(group_num integer,
        num_measurements integer, velocity_kps real, error_kps real, high_cluster integer, low_cluster integer,
        average_clearance_m real, the_geom geometry);

-- If statement to prevent the inclusion of the second of two identical rows found in the LDB (for group_num = 1)
      IF tmp_rec.group_num = 1 THEN
        IF tmp_already_included_flag = 0 THEN
          tmp_already_included_flag = 1;
        ELSE
-- If the second identical line is found, we do another fetch to skip that line
          SELECT * INTO tmp_rec FROM dblink_fetch('sediment_ss_cursor',1) AS a1(group_num integer,
            num_measurements integer, velocity_kps real, error_kps real, high_cluster integer, low_cluster integer,
            average_clearance_m real, the_geom geometry);
        END IF;
      END IF;
-- Creating a feature_id associated to this data package
      tmp_featureid = createFeature('IP',tmp_data_package_id);

-- Creation of an instantaneous_point entry.
      INSERT INTO pdb.public.instantaneous_point(feature_id, feature_code, station_id, point_type, date_time)
        VALUES (tmp_featureid, 'SEDIMENT', nextval('station_id_seq'::regclass), 1, tmp_date_time);

--  The SRID is ported from 4269 to 4326.
      tmp_geom = ST_Transform(tmp_rec.the_geom, 4326);

-- Creation of an measurement_location entry
      INSERT INTO pdb.public.measurement_location(feature_id, feature_code, geom, x_y_position_code)
        VALUES (tmp_featureid,'SEDIMENT',tmp_geom, 'unknown') RETURNING measurement_location_id
        INTO tmp_measurement_location_id;

-- Creation of PDB data (non null group_num)
      IF tmp_rec.group_num IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,803,1,1,tmp_rec.group_num,0);
      END IF;
-- Creation of PDB data (non null num_measurements)
      IF tmp_rec.num_measurements IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,804,1,1,tmp_rec.num_measurements,0);
      END IF;
-- Creation of PDB data (non null velocity_kps)
      IF tmp_rec.velocity_kps IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,805,1,1,tmp_rec.velocity_kps,0);
      END IF;
-- Creation of PDB data (non null error_kps)
      IF tmp_rec.error_kps IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,806,1,1,tmp_rec.error_kps,0);
      END IF;
-- Creation of PDB data (non null high_cluster)
      IF tmp_rec.high_cluster IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,807,1,1,tmp_rec.high_cluster,0);
      END IF;
-- Creation of PDB data (non null low_cluster)
      IF tmp_rec.low_cluster IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
```

```
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,808,1,1,tmp_rec.low_cluster,0);
      END IF;
-- Creation of PDB data (non null average_clearance_m)
      IF tmp_rec.average_clearance_m IS NOT NULL THEN
        INSERT INTO pdb.public.data(measurement_location_id, parameter_id, measuring_device_id,
          group_id, data_value, quality_flag_id)
          VALUES (tmp_measurement_location_id,809,1,1,tmp_rec.average_clearance_m,0);
      END IF;
    END LOOP;
    PERFORM dblink_close('sediment_ss_cursor');
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION insertSedimentSS() OWNER TO postgres;

-- Function that verifies that SS Sediment data are in the PDB
CREATE OR REPLACE FUNCTION verifySedimentSS()
RETURNS VOID AS
$BODY$
  DECLARE
-- Declaration of temporary variables needed for this SQL function
    tmp_rec RECORD;
  BEGIN
    RAISE NOTICE 'Verifying Scotia Shelf Sediment data ...';
-- Verification of the SS Sediment data value (parameter_id = 803 ) group_num
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 803 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT group_num FROM readbv3b.public.sediment_ss_position')
      AS t(b integer)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS group_num data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 803';
    END IF;
-- Verification of the SS Sediment data value (parameter_id = 804 ) num_measurements
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 804 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT num_measurements FROM readbv3b.public.sediment_ss_position')
      AS t(b integer)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS num_measurements data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 804';
    END IF;
-- Verification of the SS Sediment data value (parameter_id = 805 ) velocity_kps
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 805 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT velocity_kps FROM readbv3b.public.sediment_ss_position')
      AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS velocity_kps data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 805';
    END IF;
-- Verification of the SS Sediment data value (parameter_id = 806 ) error_kps
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 806 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT error_kps FROM readbv3b.public.sediment_ss_position')
      AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS error_kps data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 806';
    END IF;
-- Verification of the SS Sediment data value (parameter_id = 807 ) high_cluster
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 807 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT high_cluster FROM readbv3b.public.sediment_ss_position')
```

```
      AS t(b integer)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS high_cluster data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 807';
    END IF;
-- Verification of the SS Sediment data value (parameter_id = 808 ) low_cluster
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 808 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT low_cluster FROM readbv3b.public.sediment_ss_position')
      AS t(b integer)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS low_cluster data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 808';
    END IF;
-- Verification of the SS Sediment data value (parameter_id = 809 ) average_clearance_m
    SELECT count(*) INTO tmp_rec FROM (SELECT data_value FROM data WHERE parameter_id = 809 EXCEPT ALL SELECT b FROM
      dblink('dbname=readbv3b user=postgres','SELECT average_clearance_m FROM readbv3b.public.sediment_ss_position')
      AS t(b real)) AS t2(b2);
    IF tmp_rec.count = 0 THEN
       RAISE NOTICE 'OK (Sediment SS average_clearance_m data)';
    ELSE
       RAISE NOTICE 'Missmatch found in Sediment SS data(data_value), parameter_id = 809';
    END IF;

--    SELECT count(*) INTO tmp_rec FROM (SELECT geom FROM measurement_location WHERE feature_code = 'SEDIMENT'
--      EXCEPT ALL SELECT b FROM
--      dblink('dbname=readbv3b user=postgres','SELECT the_geom FROM readbv3b.public.sediment_ss_position')
--      AS t(b geometry)) AS t2(b2);
--    IF tmp_rec.count = 0 THEN
--       RAISE NOTICE 'OK (Sediment SS measurement_location:geom)';
--    ELSE
--       RAISE NOTICE 'Missmatch found in Sediment SS measurement_location(geom)';
--    END IF;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION verifySedimentSS() OWNER TO postgres;

-- Function which deletes the data related to SS Sediment data.
CREATE OR REPLACE FUNCTION deleteSedimentSS()
RETURNS VOID AS
 $BODY$
  BEGIN
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (group_num)...';
    DELETE FROM data WHERE parameter_id = 803;
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (num_measurements)...';
    DELETE FROM data WHERE parameter_id = 804;
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (velocity_kps)...';
    DELETE FROM data WHERE parameter_id = 805;
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (error_kps)...';
    DELETE FROM data WHERE parameter_id = 806;
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (high_cluster)...';
    DELETE FROM data WHERE parameter_id = 807;
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (low_cluster)...';
    DELETE FROM data WHERE parameter_id = 808;
    RAISE NOTICE 'Deleting data related to Scotia Shelf Sediment data (average_clearance_m)...';
    DELETE FROM data WHERE parameter_id = 809;
    RAISE NOTICE 'Deleting measurement_location related to Scotia Shelf Sediment data...';
    DELETE FROM measurement_location WHERE feature_code = 'SEDIMENT';
    RAISE NOTICE 'Deleting instantaneous_point related to Scotia Shelf Sediment data...';
    DELETE FROM instantaneous_point WHERE feature_code = 'SEDIMENT';
    RAISE NOTICE 'Deleting data_package_id related to Scotia Shelf Sediment dataset...';
```

```
    DELETE FROM data_package WHERE data_package_id = 5000001;
    RAISE NOTICE 'Done.';
  END;
$BODY$
LANGUAGE 'plpgsql';
ALTER FUNCTION deleteSedimentSS() OWNER TO postgres;
```

This page intentionally left blank.

Annex Q: user_exit_fct.sql (PL/PerlU)

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

/* This PL/PerlU function takes as inputs the decimal format of the latitude
 * and longitude of the upper left corner and the lower right corner of a rectangular
 * region defined by the user and then launches an external java code using these
 * 4 arguments. The external function will then create a PostgreSQL ETOPO2 table where
 * the columns data will be read from a ETOPO2 file and filtered using the 4 parameters
 * defined by the user.
 */

CREATE OR REPLACE FUNCTION loadEtopo2Table (decimal, decimal, decimal, decimal)
RETURNS void AS
$BODY$
  use strict;

# These are all the parameters needed for launching the external program
# IMPORTANT: These parameters MUST BE AJUSTED to your installation setup.

  my $java_cmd  = "/usr/lib/jvm/java-6-sun/bin/java"; # Full path to your Sun Java 6 application
  my $java_opts = "-cp";                              # Options passed to the java command
  my $java_pgm  = "copyETOPO2DataToPDB";              # Java program which create the ETOPO2 table
  my $jdbc_lib  = "postgresql-8.3-605.jdbc4.jar";     # PostgreSQL JDBC4 java library
  my $ncdf_lib  = "ncCore-4.1.jar";                   # ncCore java library
  my $gis_lib   = "jts-1.8.jar";                      # JTS java library
  my $inst_path = "/home/mmayrand/user_exit";         # Full path of the directory where the above
                                                      # java libraries are located

# Java Library path constructed with the above parameters which will be passed to the java program.
  my $lib_path  = $inst_path.":".$inst_path."/".$jdbc_lib.":".$inst_path."/".$ncdf_lib.":".$inst_path."/".$gis_lib ;

# These are the input parameters to be passed to the external program
# They represent the latitude and longitude of the upper left corner
# and the lower right corner of a rectangular region defined by the user
  my $lat1  = $_[0];
  my $long1 = $_[1];
  my $lat2  = $_[2];
  my $long2 = $_[3];

# Here we test if the given coordinates follow the required convention before passing them to
# the external program. If the test failed, the function prints an error and than exits.
# This convention may only be valid for the northern hemisphere and should
# be generalized to the southern hemisphere also.
  if ($lat1 < $lat2 || $long1 > $long2) {
    elog(ERROR, "\n      The two point coordinates do not follow the convention set for this function.\n
      Usage:\n
      loadEtopo2Table (upper_left_pt_lat, upper_left_pt_long, lower_right_pt_lat, upper_left_pt_lat)\n\n");
    exit;
  }

# This launches the command to the operating system as user postgres. File and library permission
# accesses should be revisited to allow proper execution of this command.
```

```
  system( $java_cmd, $java_opts, $lib_path, $java_pgm, $lat1, $long1, $lat2, $long2);

  elog(NOTICE, "Process completed for transfering ETOPO2 data to the corresponding PDB table");

$BODY$
LANGUAGE plperlu;
```

Annex R: copyETOPO2DataToPDB.java

```
/**
 * UNCLASSIFIED
 *
 * Copyright (c) Her Majesty the Queen in Right of Canada 2010
 *
 * Project: Porting of data to the Rapid Environmental Assessment Production Database (REA)
 *
 * @author: OODA Technologies inc.
 *
 * Contract number: W7707-098247
 */

// Packages needed for the compilation of this java program
// Refer to the final report for further instruction on the installation of these libraries
import ucar.nc2.NetcdfFile;
import ucar.nc2.Variable;
import ucar.ma2.*;
import java.lang.Float;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.io.*;
import java.sql.*;
import com.vividsolutions.jts.geom.*;
import com.vividsolutions.jts.io.WKTReader;

public class copyETOPO2DataToPDB {

  public static void main(String args[]) throws Exception, java.lang.NullPointerException
  {
    // Modify this line to point to your ETOPO2v2c file
    String pathToNetCDFFile = "/home/mmayrand/user_exit/ETOPO2v2c_f4.nc";

    // Initialization of local variables
    int sub_dim_x = 0;
    int sub_dim_y = 0;
    ArrayFloat.D1 sub_x = null;
    ArrayFloat.D1 sub_y = null;
    ArrayFloat.D2 depth = null;
    NetcdfFile dataFile = null;

    // Converting the 4 input parameters into float type numbers inside an array.
    final float coord[] = new float[4];
    for (int i = 0; i < 4; i++) {
      if (args[i] != null ) {
        coord[i] = Float.valueOf(args[i].trim()).floatValue();
      }
    }

    try {
      // Opening the NetCD file.
      dataFile = NetcdfFile.open(pathToNetCDFFile, null);

      // Locating the three coordinates define in the NetCDF file
      Variable dataLongVar  = dataFile.findVariable("x");
      Variable dataLatVar   = dataFile.findVariable("y");
      Variable dataDepthVar = dataFile.findVariable("z");

      // Testing if one of the three coordinates is missing.
      // If it is the case, an error is return and the program is stopped.
      if (dataLongVar == null || dataLatVar == null || dataDepthVar == null) {
```

```
  System.out.println("Cant find Variable data");
  return;
}

// Extracting the dimensions of the full grid of the file
int dimX = dataLongVar.getDimension(0).getLength();
int dimY = dataLatVar.getDimension(0).getLength();

// Initialization of the parameters of a variable window that will be resized
// to the user defined rectangle
int[] shape = dataDepthVar.getShape();
int[] origin = new int[1];
int[] sp = new int[1];
sp[0] = 1;
ArrayInt.D2 dataArray;

// Checking that the initial shape has the same dimension as the X and Y header values.
assert shape[0] == dimX;
assert shape[1] == dimY;

// Temporary counter initialization and array declarations
int cntX = 0;
int cntY = 0;
ArrayFloat.D1  tmpx = new ArrayFloat.D1(1);
ArrayFloat.D1  tmpy = new ArrayFloat.D1(1);
ArrayFloat.D2  tmpz = new ArrayFloat.D2(1,1);

// Debugging statement: System.out.println("Dim x = "+dimX+" Dim y = "+dimY);

// Calculating the dimension of the sub grid from the user defined rectangle
for (int j=0; j<dimX; j++) {
  origin[0] = j;
  tmpx= (ArrayFloat.D1) dataLongVar.read(origin, sp);
  if ( coord[1] < tmpx.get(0) &&  tmpx.get(0) < coord[3] ) {
    cntX = cntX + 1;
  }
}
for (int i=0; i<dimY; i++) {
  origin[0] = i;
  tmpy = (ArrayFloat.D1) dataLatVar.read(origin, sp);
  if ( coord[0] > tmpy.get(0) &&  tmpy.get(0) > coord[2] ) {
    cntY = cntY + 1;
  }
}

// Initialization of the arrays that will received the extracted data
// using the above dimensions calculated from the user defined coordinates.
sub_dim_x = cntX;
sub_dim_y = cntY;
sub_x = new ArrayFloat.D1(sub_dim_x);
sub_y = new ArrayFloat.D1(sub_dim_y);
cntX = 0;
cntY = 0;

// The X subset (longitude) is transfered in the sub_x array
for (int j=0; j<dimX; j++) {
  origin[0] = j;
  tmpx= (ArrayFloat.D1) dataLongVar.read(origin, sp);
  if ( coord[1] < tmpx.get(0) &&  tmpx.get(0) < coord[3] ) {
    sub_x.set(cntX,tmpx.get(0));
    cntX = cntX + 1;
  }
}
// The Y subset (latitude) is transfered in the sub_y array
```

```
  for (int i=0; i<dimY; i++) {
    origin[0] = i;
    tmpy = (ArrayFloat.D1) dataLatVar.read(origin, sp);
    if ( coord[0] > tmpy.get(0) &&  tmpy.get(0) > coord[2] ) {
      sub_y.set(cntY,tmpy.get(0));
      cntY = cntY+1;
    }
  }

  // Initialization of the 2D array for the depth data
  depth = new ArrayFloat.D2(sub_dim_y,sub_dim_x);

  // Initialization of parameters needed for the final extraction
  // of the depths associated to the subset defined by the user.
  cntX = 0;
  cntY = 0;
  int[] originx = new int[1];
  int[] originy = new int[1];
  int[] originz = new int[2];
  int[] spz = new int[2];
  spz[0] = 1;
  spz[1] = 1;

  // The Z (depth) subset (defined by the user) is transfered in the 2D depth array,
  // any depth values outside the user defined rectangle (coord) is disregarded
  for (int i=0; i<dimX; i++) {
  originx[0] = i;
  tmpx= (ArrayFloat.D1) dataLongVar.read(originx, sp);
    if (coord[1] < tmpx.get(0) &&  tmpx.get(0) < coord[3]) {
      for (int j=0; j< dimY; j++) {
        originy[0] = j;
        tmpy = (ArrayFloat.D1) dataLatVar.read(originy, sp);
        if ( coord[0] > tmpy.get(0) &&  tmpy.get(0) > coord[2] ) {
          origin[0] = sub_dim_y*i+j;
          originz[0] = cntY; originz[1] = cntX;
          tmpz = (ArrayFloat.D2) dataDepthVar.read(originz, spz);
          depth.set(cntY,cntX,tmpz.get(0,0));
          cntY = (cntY+1)%sub_dim_y;
        }
      }
      cntX = cntX + 1;
    }
  }

  // Debugging statement: System.out.println("dim z "+dataDepthVar.getSize());

  // try/catch block for testing the proper reading of the file before closing it
} catch (java.io.IOException e) {
    e.printStackTrace();
    return;
} catch (InvalidRangeException e) {
     e.printStackTrace();
} finally {
  if (dataFile != null) {
    try {
        dataFile.close();
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
  }
}

Connection con = null;   // JDBC Connection initialization
```

```
    // In this try/catch section, we open a JDBC connection to transfer the selected data
    // to the ETOPO2 table of the PostgreSQL PDB database
    try {
      // We open the JDBC connection to the PDB database
      // Adjust the below parameters to your installation settings if needed
      String url = "jdbc:postgresql:pdb";
      String userName="postgres";
      String password="";
      Class.forName("org.postgresql.Driver");
      Connection conn = DriverManager.getConnection (url, userName, password);
      //Debugging statement: System.out.println ("Database connection established");

      // We first empty the ETOPO2 table from the PostgreSQL PDB database
      // without removing the structure of the ETOPO2 table
      Statement sta = conn.createStatement();
      sta.executeUpdate( "DELETE FROM etopo2");

      // Creating a PreparedStatement for inserting new records
      PreparedStatement ps = conn.prepareStatement(
        "INSERT INTO etopo2 (geom, depth) VALUES (CAST(? AS geometry),?)");

      // The following statements transfer the above selected ETOPO2 values to the ETOPO2 table
      // The ETOPO2 values is converted into a 3D GIS geometry using the SRID 4326
      PrecisionModel pm = new PrecisionModel(PrecisionModel.FLOATING);
      GeometryFactory geometryFactory = new GeometryFactory(pm,4326);
      WKTReader geometryReader = new WKTReader(geometryFactory);
      // Debugging statement: System.err.println("dim x="+sub_dim_x+" dim y"+sub_dim_y);
      for (int i = 0; i < sub_dim_x; i++ ) {
        for (int j = 0; j < sub_dim_y; j++ ) {
          // We generate a geometry based on the lat/long for each point of the grid
          Geometry geom = geometryReader.read("POINT ("+sub_x.get(i)+" "+sub_y.get(j)+")");
          // Debugging statement: System.err.println("POINT ("+sub_x.get(i)+" "+sub_y.get(j)+")");
          // The following settings will fill the template SQL statement defined above as a PreparedStatement
          ps.setString(1, geom.toString());
          ps.setFloat(2, depth.get(j,i));
          // The SQL statement is sent to the PDB database
          ps.executeUpdate();
        }
      }
      System.out.println( "*** Transfer from the ETOPO2v2c NetCDF file to PosgreSQL completed.");
      ps.close();
      sta.close();
      conn.close();
    } catch (Exception e) {
        System.err.println("Exception: "+e.getMessage());
        e.printStackTrace();
    }
  }
}
```

List of symbols/abbreviations/acronyms/initialisms

DB	Database
DRDC	Defence Research and Development Canada
ECSWAN	Eastern Canadian Shallow Water ambient Noise
GFI	Government Furnished Information
GUI	Graphical User Interface
HP	hydrophone
ID	identifier
LDB	Load database
NADAS	Non Acoustic Data Acquisition System
PDB	Production database
REA	Rapid environmental assessment
SS	Scotian Shelf
SWDB	Shallow Water Data Base
TSD	Temperature/Salinity Dalhousie dataset
XBT	eXpendable BathyThermograph
XSV	eXpendable Sound Velocimeter

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

OODA Technologies Inc.
4891 Av. Grosvenor, Montreal Qc, H3W 2M2

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED
(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC JUNE 2010

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Implementation of the rapid environmental assessment production database

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Mayrand, M.

5. DATE OF PUBLICATION (Month and year of publication of document.)

October 2010

6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)

132

6b. NO. OF REFS (Total cited in document.)

2

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Atlantic
P.O. Box 1012, Dartmouth, Nova Scotia, Canada B2Y 3Z7

9a. PROJECT NO. (The applicable research and development project number under which the document was written. Please specify whether project or grant.)

Project 11CH

9b. GRANT OR CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

W7707-098247/001/HAL

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Atlantic CR 2010-123

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This contract deals with the construction of the redesigned Rapid Environmental Assessment (REA) database. The data model, as described in the report *Utilizing Arc Marine concepts for designing a geospatially enabled database to support rapid environmental assessment* [1] was implemented. SQL scripts were constructed for porting the data from the initial READBv3b to the new REA Production database. A user interface for executing these SQL scripts was also created. Functionality to validate the porting process was also included. A user-exit was created for accessing the ETOPO2 bathymetric data located in an external NetCDF file. This report contains the details of the porting process, all the instructions for using the user interface, the SQL scripts code, and finally a list of recommendations and possible tasks for future improvements.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Database
PostgreSQL
Rapid Environmental Assessment
REA

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE DÉFENSE

www.drdc-rddc.gc.ca