|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date Reviewed** | **Reviewed by** | **Checked by** | **Description** |
| 0.1 | 21-02-2017 | DaMcL |  | Insert ISO19030 Flowchart, describe interface and use cases. |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Change Record**

# Hull Performance Database and Interface

## Introduction

This document describes an existing MySQL database and associated MATLAB codes for organising, processing and analysis of vessel performance data.

The overall structure of the database and codes is first explained. Then, the interface to the system is explained. Finally a number of use cases are described.

It is recommended to check the meaning of terms in the glossary before continuing.

## Database Structure

The database stores performance data (and KPI’s calculated from this) processed by a number of different agencies. These agencies can be external companies who process the data for us but the processing can also be done internally and this is carried out by the database itself through a stored procedure. In this case, the performance data is processed from data retrieved from sensors on-board the vessels and this data is referred to as raw data. The database also stores this raw data.

The raw data can come from both the vessel owners and the external companies (the external companies provide processed data but will also provide the raw data from which this was calculated). In either case, data relevant to the process is extracted from the data received and inserted into the raw data. In the former case, the data received can take any form (vessel owners are under no obligation to organise their data in a consistent way) and in the latter, it is structured the same for all vessels but is different for the different companies. The received data from each external company is stored in the database.

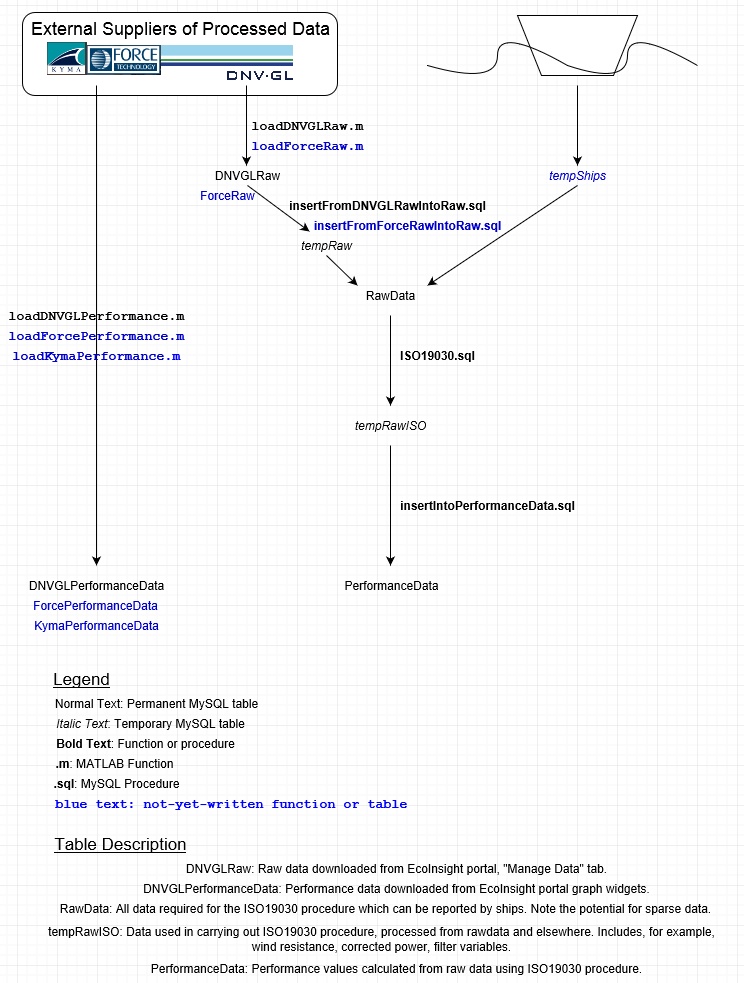


Figure . Flowchart for ISO19030 process, showing data flow.

Based on these requirements for storage, a number of requirements for processing arise and these are illustrated in Figure 1. Note that there is a simple flow from the external companies to the corresponding processed data table through the arrow at the left of the figure. In this case, a method is required to load the data from the file to the corresponding table, these methods are named similarly to the tables which they load into.

In the rest of the figure, raw data is loaded from the received file and then modified to fit the format specified by the raw data table. In the case of a received file originating from DNVGL, the file would first be loaded into the DNVGLRaw table using the loadDNVGLRaw.m function. Note that a single file can contain data from multiple vessels. The next step is to extract from DNVGLRaw the data relevant to the processing and add it to the RawData table after converting the units of some of the columns, changing conventions and simply relating the names of columns in one table to columns in the other. This, and subsequent steps, are performed for one vessel at a time.

Now that the data is in the Raw Data table, we can call the procedure for calculating the processed data, called ISO19030.sql. This procedure creates a table, tempRawISO, containing the data for the selected ship. Then it makes many calls to other procedures, updating the values of tempRawISO so that the performance data can be calculated. When that is done, the final step is the procedure insertIntoPerformanceData.sql.

This function exists to apply a set of filters prior to the data being inserted into the PerformanceData table because the choice of these filters must be made by a human. After the data has been checked, the procedure is called and the processed performance data for this vessel are finally inserted into PerformanceData.

The remaining and final step is the calculation of the KPI’s from this performance data. This has not yet been implemented but will require the creation of some new tables which will be much smaller than all those described above.

Other tables, apart from those mentioned above, are required for the ISO19030 procedure to work. For example, the table “Vessels” describes each vessel for which some raw or processed data is available, with columns including IMO Vessel Number, breadth moulded etc. The table SpeedPower contains the speed and power values on which the processing is based, i.e. columns for speed, power, displacement, trim and the corresponding IMO Vessel Number.

Note that not all functions in Figure 1 are already written. Currently, the structure is in place for raw and performance data sourced from DNVGL only.

## Interface

The system can be interacted with in a number of ways. It was mentioned in the section Database Structure that some of the functions are MATLAB functions while others are MySQL stored procedures. This has arisen because MySQL procedures alone couldn’t provide sufficient automation. An example of this is that a MySQL procedure cannot be used to take as input a file path and then load the contents of that file into a specified table. Neither file names nor table names can be passed as variables to the MySQL “LOAD INFILE” command. The nearest alternative would be replacing the text in a script corresponding to these values. But a MATLAB function can do this by taking as inputs the table name and file name and generating a string containing the appropriate MySQL syntax, then executing that syntax on the MySQL server.

Unfortunately, there is no clear boundary between the MATLAB and MySQL parts of the system. In general, MySQL procedures are used wherever possible and MATLAB otherwise. However, MATLAB can be used to call MySQL and not vice versa. Therefore it is currently imagined that the entire process will be accessed through MATLAB.

The MATLAB code consists of classes which represent the vessels and some relevant parts, e.g. cVessel, cVesselEngine, cVesselWindCoefficient. The first of these, cVessel, inherits from the others. When the user wants to write data to the database, he creates a cVessel object and assigns static ship data to its properties based on what he received from the vessel owner. This object will then insert the data from these properties into the appropriate tables in the database (e.g. the “Vessels” table will be updated with static ship data).

The data insertion is handled by an inherited class named cMySQL. This class creates strings containing the MySQL syntax and executes the commands on the MySQL server. Methods of this class include insertValues, select, update, all of which will execute the corresponding SQL syntax on a user-selected table, column, values etc.

To load the time-series data, the class cHullPerDB is used and currently only has the methods shown in Figure 1 loadDNVGLPerformance and loadDNVGLRaw.

Reading out of data and analysis is done with the cVesselAnalysis class. When the user wants to get some data from the database he instantiates an object of this class with the input IMO vessel numbers and, optionally, values specifying which dry docking intervals are of interest. The result is an array of cVesselAnalysis objects with as many rows as the number of IMO as many columns as the number of dry docking intervals. An example of a method of cVesselAnalysis is plotPerformanceData which generates plots used in the data analysis. This method will use data returned by other cVesselAnalysis methods such as regressions and moving averages and enhance the plot with this data.

## Use Cases

### Case 1: Load Performance Data, generate plots for reports

The user has retrieved files from an external company containing performance data for a number of vessels belonging to one vessel owner. The customer expects a report on the performance and so one step is to plot the data and illustrate the trend lines, moving averages etc. The first step is to load the data into the appropriate table.

He starts by instantiating the appropriate object, in this case a cHullPerDB. Say the external company is DNVGL. Then the appropriate method will be loadDNVGLPerformance for this type of file. He passes in the file paths in a cell array along with the corresponding IMO vessel numbers (these are not stored in the files). The result is that all the data for the vessels is now in the appropriate table, in this case DNVGLPerformanceData. Had any data already been in this table for the same IMO and time, then the other columns at the same rows would have been updated with these new values.

The next step is to create a cVesselAnalysis object with the appropriate IMO numbers. The output will be an array containing all the data previously added for these vessels. So a call to the methods regressions and movingAverages on the array will result in regression coefficients and values of moving averages over the selected duration being assigned to the appropriate properties of each element of the array. Now a call to method plotPerformanceData on the array will result in plots, one for each vessel, of the data read from the database, overlaid with linear trendlines for each dry-docking interval and horizontal bars representing moving averages of the selected duration.

### Case 2: Load raw data, process performance data, analyse

The user has raw data from a vessel owner and the owner has requested a vessel performance analysis be carried out for a number of vessels, so he needs graphs and other data such as KPI’s trend-line coefficients.

Currently, raw data can only be loaded for files sourced from DNVGL, so the user will forward the raw data files to them and receive a “DNVGL Raw” file in return, i.e. a consistently formatted file containing (essentially) the same data. This file contains data for multiple ships so the user will create an object of the cHullPerDB class and call the method loadDNVGLRaw on this file. This method will call inherited cMySQL method loadInFileDuplicate which executes the SQL code to load the data into the appropriate table, DNVGLRaw, without duplicates.

Before raw data can be processed into performance data, the static ship data is required. This can be loaded into the database before or after the raw data. The user creates a cVessel object and assigns the appropriate data to its properties (e.g. data for speed and power, ship dimensions, wind coefficients etc). Then the user calls the appropriate “insert\_” method of the object which will insert the data via the cMySQL inherited class, into the appropriate table.

Now at this point, MySQL procedures must be called directly because no MATLAB functions have been written yet to call them. So the user calls insertFromDNVGLRawIntoRaw with the IMO of the first vessel, followed by ISO19030 with the same IMO again as input. The first of these adds that vessel’s relevant data to table Raw Data. The second creates a table called tempRawISO, inserts the data for that vessel into it from Raw Data, and processes this data. Now the data is processed in the table tempRawISO but not added to the final table. So the user must inspect the data to decide which filters to apply and then call insertIntoPerformanceData with the chosen filters as inputs. Now, the data for this ship is in the table PerformanceData. So all the steps in this paragraph now must be repeated for the remaining ships (this is why we use MATLAB).

To analyse the data from the table PerformanceData, the user instantiates MATLAB objects of the class cVesselAnalysis, with the input being an array of the IMO vessel numbers. The output is an array of objects with as many rows as IMO numbers and as many columns as dry docking intervals. Now the user runs the methods regressions and movingAverages to get the coefficients and average values, then plotPerformanceData will generate these plots for all vessels.

## Glossary

|  |  |
| --- | --- |
| *Received data* | Data received from vessel owners or others, in its original form. |
| *Raw data* | Data relevant to the processing of performance values. |
| *Performance value* | A time-dependent measure of the performance of the vessel hull and propeller. |
| *KPI* | Any of a number of key performance indicators calculated from the performance values, which are generally scalar values taken over a time-interval (e.g. an average). |
| *Static ship data* | Data for a vessel which is not considered to change with time (e.g. breadth moulded, length overall). |
| *Time-series data* | Data which is dependent on time (in this context, either raw data or performance data). |
| *IMO vessel number* | A unique identifier for vessels issued by the International Maritime Organisation. |
| *Dry Docking* | The process of taking a vessel into dry dock, i.e. a dock in which the level of water is controlled, for maintenance, repainting and other activities. |
| *Dry Docking Interval* | The duration between two successive dry dockings for a vessel. |