491 | Februar 1989

SCHRIFTENREIHE SCHIFFBAU

E. Kantorowitz, A. Birbanescu-Biran und J. Yanai

Database-Oriented Ship Design

TUHH
Technische Universität Hamburg-Harburg

Database-Oriented Ship Design

E. Kantorowitz, A. Birbanescu-Biran, J. Yanai Hamburg, Technische Universität Hamburg-Harburg, 1989

Schriftenreihe Schiffbau
Schwarzenbergstraße 95c
D-21073 Hamburg

http://www.tuhh.de/vss

INSTITUT FÜR SCHIFFBAU DER UNIVERSITÄT HAMBURG

Bericht Nr. 491

DATABASE-ORIENTED SHIP DESIGN

von

E. Kantorowitz
A. Birbanescu-Biran
J. Yanai

Februar 1989

ISBN 3 - 89220 - 491 - 8

Institut für Schiffbau
Universität Hamburg
Lämmersieth 90
D-2000 Hamburg 60

Table of contents

Abstract

The successful automation of naval architectural computations involving different programs depends on the uniformity and flexibility of the data management. In attempt to achieve these desiderata, a system based on a normalized relational database, containing neither multiple occurrences of the same data item nor data derived by calculations, was implemented. Potential sources of errors and manual handling of data and programs were systematically removed. A method was employed for running a sequence of several programs by one command. In some cases, ship-analysis time was reduced from hours to minutes.

Introduction

The computer programs available to the naval architect are often of different origins, necessitating the mastering of their different requirements. The user must be very careful to follow their rules, since any minor deviation may corrupt the results of the computations. Preparation of the input data for a number of programs is, therefore, a cumbersome procedure that both consumes time and involves ample possibilities for human errors. The following aspects of the process seem especially disturbing.

- The same data must sometimes be entered into different programs in different ways. One program may require that the stations be defined by their distance from AP, while another measures the distances from FP.

- The same data item has to be stored several times, once for each program that needs it. Whenever such data are changed during the design process, the naval architect must be careful to update every file in which these items appear. Since each file stores data in its own particular way, the updating process is tedious and provides opportunities for human error.

- The output of one program cannot be used as input to another program. Thus, several hours work may be required to convert the results of a computation into input for another ship-analysis program. For the authors of this paper it was a depressing feeling to work for hours with hand-held calculators in order to obtain a few seconds of mainframe computations.

- The possibility exists of obtaining results that correspond to data that are no longer valid. This may occur with programs that store some intermediate results in order to save computations. One example concerns the tank capacities stored by one program for later usage; however, if the position of a bulkhead were changed during the design process, the stored capacities would no longer be correct. The problem was that the capacities were derived from an older version of the input data. This may be called the *derived data* problem. Errors can be avoided if the derived data are not stored at all, but instead are computed from the current version of the input data every time they are needed.

- The data of the ships are dispersed among the input files of different programs. Because the structures of these files are tailored to the specific needs of the particular programs, they are not suitable for any other application. It is a waste that the ship information collected in these files, at high expense, cannot be exploited for any other purposes. Such a database could be used, for instance, for statistical studies to predict the properties of new ships. Ideally all information should be stored in a way that it is readily applicable to any program.

The problems described are related to the way in which data are managed in traditional programs. More flexible methods for data management are obviously required. Two decades ago, it was realized that some of the difficulties may be alleviated when all the different ship programs are designed to use the same database on the computer. Data modifications could then be done only to this single database. The problem of keeping databases of different programs consistent with one another would no longer exist because there would be but one database. In systems of this kind, all the programs are designed from the beginning to work with the same input files. An example is a system written at the University of Hanover [1], called ARCHIMEDES, in which [illegible] of stability, damage and longitudinal bending calcula-

In later commercial ship design systems, such as FORAN [2] or NAPA [3], a large number of programs are integrated around a number of common databases. Another interesting method of integrating several ship-design programs in one system is SPIRAL, developed at the University of Michigan [4], [5]. Adaptation of an existing program to the SPIRAL system, the rules for which are published, requires some not always trivial program changes. Programs integrated into SPIRAL employ their own special input data files, and there is no central database. The major service of the system is to provide a convenient, standardized user interface to all the programs.

While naval architects were busy developing their programming systems, computer scientists were engaged in basic database research. One of their goals was to achieve *data independence;* that is to make the application programs independent of the way in which data are stored in the computer. The information stored in the database may then be exploited by any program using a convenient, high-level *query language.* Another goal was to ensure the consistency of the database under different situations, including system failures. The idea was to solve these intricate problems with a special piece of software, called the *database management system,* or *DBMS.* To achieve these goals of data independence and consistency, different kinds of *data models* were investigated. Early DBMS's were based on the hierarchical model, which assumed a hierarchical relationship between the different components of the database. In the early seventies, interest shifted to the more general *network* model. Satisfactory solutions to the major database problems were achieved in the late seventies with the *relational model.* This model is intuitive for the user because the database is stored in a number of tables that implement the mathematical concept of *relations.*

For the relational data model, a solid mathematical theory has been developed together with sound practical methods for the construction of such database systems. The relational database technology has by now matured, and a number of systems are commercially available. The interface between these systems and the application programs may be written in the standard SQL language. An application program that uses the services of such a system may be written in any popular programming language. For example, if FORTRAN is used, the SQL statements are inserted in the FORTRAN code in those places where DBMS services are required. It is claimed that such a FORTRAN program may work with any DBMS that meets the SQL standard. The theory and practice of relational data systems are discussed in textbooks, such as Ullman [6], Delobel and Adiba [7], Vinek, Rennert and Tjoa [8], Fargette [9], Date [10] and Korth and Silberschatz [11].

Vendors of some ship CAD systems claim that these systems are based on a relational database. Xia [12], however, criticizes the AUTOKON, FORAN and HCS systems for maintaining redundant databases, for having no provisions for enforcing data consistency and for having no mechanisms for interactive interfacing. He proposes a system called CRDB, the Chinese Relational Data Base System, which does not have these disadvantages.

Allieri and Sartori [13] analyze various data models and reach the conclusion that the hierarchical and network models do not answer all the needs of a ship-design system. They developed a relational database for CETENA, in Italy.

The Computer Supported Design (CSD) project, described in [14], is based on the ship "product model." This concept is represented by a database that defines the ship and all its components in full detail. The database is to be stored in a CAD-type database system using a hierarchical model. Such a system is expected to be especially suitable for geometric modeling, drafting and manufacturing control. From this database, the data required for ship analysis will be extracted into a relational database, which will be employed by the analysis programs.

The present paper describes a system based on the principles sketched in [15] and further elaborated in [16]. Progresses and experiences with the system are reported in [17],[18],[19],[20],[21], [22] and [23]. The system, developed over a number of years at the Technion, in Haifa, was originally intended for supporting ship analysis and design. The major design goals were as follows:

- Enable the ship designer to accomplish complex computations involving several programs and much data with a minimum of effort.

- Ensure with all practical means the correctness of the data employed in the calculation; with large amounts of data from different sources, the assurance of the correctness of the data was regarded as a central issue.

- Organize the database such that the information stored in it may be readily exploited by any current or future program.

- Permit the simple integration of existing programs into the system; programs that have been used for many years are very valuable, in that there is a considerable body of experience as to their applicability, correctness and accuracy.

- Allow operation of all the programs of the system in the same way so that the user has little to learn.

The result of this research is a computer-aided ship design system integrated around a relational database that essentially contains a detailed description of the ship and its components. Each data item is stored in only one place. An update of the item will therefore be done in this place, which will at any time contain the current item value. Any program that needs the item will access this place and, therefore, will always obtain the current value. The database is normalized in a relational sense (see references [6] to [11]), and the information stored in it is therefore readily applicable to any future program. The system also enables naval architects to perform with a single command complex calculations involving several programs. The instructions required to activate the programs involved are stored in a file of commands, which also contains statements that activate special *interface programs* for retrieving the required data from the data base and accomplishing the required data conversions.

In the present paper, the main design ideas of the system are discussed first, followed by the ship engineering considerations that were employed for the design of the database. The techniques enabling a sequence of computations to be executed completely automatically are outlined, and a number of applications demonstrating the power of the developed system presented. The simplicity and flexibility of the system are primarily due to the use of a single data standard for the entire system. All the programs employed were thus adapted to meet the standard. Adoption of such a standard by the ship industry would facilitate the exchange of ship data and programs as well as a better utilization of these resources. It may also facilitate cooperation among designers, constructors and operators of ships, who will be able to exchange well-defined ship data that can be readily processed by their computers. The detailed specification of the data standard employed in the Technion system, as given in the appendix, may serve as a model for such a standard.

The principles employed in the TECHNION system

The ship design system developed at the TECHNION is organized around a single, comprehensive engineering database, which is employed for both ship analysis and configuration control. This database, therefore, contains all the data required for defining a ship. The database is stored in a number of tables. The division of the data into the various tables was based on engineering considerations: i.e., each table stores all the information regarding a particular aspect of a ship component. The tables show the relationships between these entities and are called *relations*. The relations designed at the TECHNION appear in Fig. 1, which actually shows more relations than presently employed. The chart contains most of the relations that we believe to be necessary for a ship design office. A detailed discussion of the groups of relations shown in Fig. 1 is found below,in the section entitled "The Database."

A comprehensive ship design system will include different programs to cover every kind of calculation that may be needed. It must also support all kinds of data required in ship design. In order to make such an inherently complex system manageable for both programmers and users, significant simplifications had to be introduced. One way of achieving this was to employ a number of standards that are valid all over the system. The principles of the standards adopted for the TECHNION system will be now discussed.

- A single data standard is employed for all application programs in the system. All ship data are stored in a standardized set of tables, the structures of which are defined in the appendix. Each kind of ship data is stored in a particular column in one of these tables, and each table will typically be stored in a separate computer file.

- Whenever practical, data are stored in the system in SI units. This is a simple rule, which is easy to explain and remember. It may be argued that these units are not always appropriate. Users, however, will normally not look into database tables but will work with programs that employ these tables. Such programs will probably display the results of their calculations in practical units, like hp and knots. Programs may also request the user to supply data in these units. One advantage of employing SI units is that they enable a useful program-writing convention: i.e., programs should internally store only data expressed in SI units and employ clean, homogeneous physical formulas. Conversion to units like hp or knots is made only at the moment that the results are to be displayed to the user. Similarily when data expressed in other units than SI are entered by the user, the program will immediately convert them to SI units and store them in these units. This simple programming convention will reduce possibilities that the programmer will make errors with the units.

- All the database tables are processed with the same single set of tools. An example of such a tool is a facility for entering data into the tables. The same single facility is employed for entering data in any of the tables of the system. The result is that the naval architect has to learn the usage of only one set of tools .

- A standard method is employed for integrating old existing programs into the TECHNION system. The difficulty in doing so is that each program assumes that its input data are organized in a particular way. The standard TECHNION solution consists in writing a special *interface program* that retrieves the required input data from the standard TECHNION database tables and stores them in an *intermediate file* in the format required by the particular program. The retrieval of the required data and the creation of the intermediate file take places every time the program is activated. This ensures that the program always employs up-to-date data.

- When data are entered into the database, their correctness is checked by a set of *integrity constraints;* for instance, the breadth of the ship is less than its length.

- Any ship data item -- e.g. the breadth of the ship -- is stored in only one place in the database tables. If such an item is to be modified, this change has to be done only in this one place. This principle of having only one copy of each data item ensures that when different programs retrieve a data item from the database, they will obtain the same value, and that this value is up-to-date. The implementation of the principle is quite simple. The tables of the database are defined in such a way that data redundancy is not possible. The database is *normalized* in the relational database theory sense.

- The standard database stores only *primary data*, that is, data that may not be derived from any other data of the database by a known practical method of computation. *Derived data* -- that is, data that may be so derived -- are not stored in the database. An example of derived data is a ship's displacement, which may be computed by adding the masses of all the weight items. Since the displacement value is not stored in the database, it must be calculated for any program that needs it. This ensures that calculations are always done with the displacement that corresponds to the current ship configuration. The price of this principle is that the value must be computed anew each time when needed. With today's fast computers, however, this is usually an acceptable price.

The employed database schema is proposed as a model for establishing a standard for ship data and programs that will enable the sharing of databases, and thus facilitate cooperation in ship design, construction and operation.

The Database

The database developed at the TECHNION contains information about sets of objects of interest to ship engineering; for example ships, hulls, engines, propellers and bulk-cargo materials. The data of these objects are stored in tables designed in accordance with the relational database theory (see, for example, references [6] to [11]). These tables represent *relations*, a term that comes from set theory. The values that appear in a particular table column represent one of the *attributes* of the object and must be taken from a specified set of values, called its *domain*. For instance, the length between perpendiculars, in meters, 228.000, is an attribute of the ship *Amanda Miller. This value, 228.000, must belong to the domain LENGTH*. In this case, domain contains positive numbers that represent lengths. The rows of the tables are called *tuples*, each of which is a sequence of *attribute* values characterizing a database object. No duplicate tuples are allowed in a relation. Each tuple is uniquely identified by a *key*, which is composed of one or more attributes. For instance, in the relation termed **STATNX,** which defines the longitudinal position of ship stations, the key is the ordered pair (ship identifier, station identifier).

The database includes a *data dictionary*, which contains a characterization of all the data stored in it and is kept in a number of relations. The TECHNION system has a relation called **DDCOLS** (data dictionary - columns) that stores a specification of all the columns of every table in the database. A column specification includes its name, FORTRAN format and the domain to which it belongs. Another relation, called **DDOMAIN** (data dictionary - domains), stores the specifications of the domains. One of its attributes is the unit used for data belonging to the particular domain. Usually the units used for storage belong to the SI international system ; if other units are employed to display the data, DDOMAIN contains the appropriate conversion factors. For example, angles, such as *trim angles*, are always stored in units of radians. In computer programs, however, it may be more convenient both to input and to output angles in sexadecimal degrees. The corresponding conversion factors are the numerical values of $\pi/180$ and $180/\pi$. Finally, the data dictionary may be read by both human users and computer programs. Thus, a computer program looking for a certain attribute value may automatically find the relation, the format and the units in which the attribute is stored.

Fig. 1 shows the way in which data are partitioned into relations in the TECHNION DB. The functional hierarchy of this partition is explained by the *tree graph* appearing at the top. The branches of this tree continue downwards, and the various relations are shown as tree *leaves*. Along each branch, the relations may appear at three levels, which express the "generality" of the data. Relations shown at the upper level contain data of particular ships or hulls. Relations appearing at the second level describe data that may be employed in several different ships; for example, engines and propellers. These relations are actually "catalogs" and may be called *Master Equipment Lists*, or *MEL*. The relations appearing at the lowest level contain general engineering data; for instance, densities and kinematic viscosities of water at different temperatures. These data were termed *library data*, and their inclusion contributes toward making the database self-contained as well as simplifies automatic program execution. If, for example, tables of water densities are stored in the DB, there is no need to consult engineering handbooks or tables of physical constants when preparing the input for a program requiring such data. What remains for the user to do is to specify whether salt or fresh water is to be considered and at which temperature. Programs that need these data have to retrieve the appropriate density values by themselves and, if necessary, interpolate between them.

To summarize, in Fig. 1 relations are organized along vertical branches according to a functional criterion; advancing downwards along the same branch, the data go from the particular to the more general. Data-dictionary relations are not shown in Fig. 1.

A general description of the database was given in this chapter, together with some of the considerations that led to the schema adopted. Besides these considerations an attempt was made to define *normalized* relations, in the sense described in references [6] to [11]. The purpose of *normalization* is to avoid *data*

redundancy as well as problems that can arise when adding, updating or deleting data. The principles of normalization are explained in [21], with examples relevant to ship design. Most of the relations appearing in Fig. 1 are detailed in the Appendix.

GENERAL SHIP DATA

In the TECHNION DB a distinction is made between *ship* and *hull.* By *hull* a particular hull *form* is meant. The general data of such a hull are stored in relation **HULL.** The key of this relation is the hull number, called HULLNO, which uniquely defines the particular lines within the DB. Other attributes in the same relation are the main dimensions, L, B, T, the hull origin (hull series or designer) and the hull identification within its series.

By *ship* we understand a *ship realization,* that is an actually designed (at least at the preliminary design stage) or built ship for which main dimensions are available. The general ship data are stored in relation **SHIPS.** The key attribute of this relation is the ship identifier, called SHIPNO, which is unique throughout the database. The hull shape of this ship is specified by the value of the attribute HULLNO.

Several ship designs may be based on the same hull. In the simplest case sister ships belonging to the same series will have exactly the same hull form. More generally, several ships may be variants of the same hull differing in main dimensions, appendage configuration, general arrangement and/or internal subdivision. Moreover, from the same hull form various ships can be generated by *affine* transformations in which one or both L/B and B/T ratios are changed, but not coefficients of form such as C_B or C_M.

For the reasons described above SHIPNO may be different from HULLNO. However, it is recommended that HULLNO be equal to the SHIPNO value of the first ship in its series. Sister ships will thus have different SHIPNO values but the same HULLNO. For example, three sister ships may have the SHIPNO values 100, 101 and 102, respectively. For all three vessels HULLNO = 100.

Further attributes of the relation **SHIPS** are the main ship dimensions, the ship name, flag and type, the design or nominal displacement (used for reference purposes) and speed and range performances.

The ratios of the principal dimensions given in relation **SHIPS** for a given SHIPNO value, to those stored in relation **HULL** for the corresponding HULLNO value, define the *affine transformation scales* to be used in hydrostatic calculations or the *geometrical similitude scale* to be used in resistance and powering calculations. This scaling may be used for the derivation of new designs, from a parent ship or hull defined in the database.

Two library-data relations are shown on the branch of general ship data in Fig. 1. These are **FLAGS** and **TYPES.** The former stores abbreviations of country or owner names, the latter acronyms of ship types. These library data are used as attribute values in relation **SHIPS.**

HULL GEOMETRY

This part of the database contains the definition of the hull surface, the side and transversal contours of the ship as well as her internal subdivision.

The hull surface is defined in the traditional way by a number of transversal sections. A hull can be defined as one solid or it can be divided into several solids in order to facilitate computations. In our system these solids are called *longitudinal spaces*. The concept of *longitudinal space*, as used here, originates from the program ARCHIMEDES [1]. The general characteristics of longitudinal spaces are defined in relation **LSPACES.** If a ship hull is defined by several longitudinal spaces, then relation **LSPACES** contains a tuple for each one of them. The key of this relation is composed of two attributes, the hull number, HULLNO, and the space number, SPACEN. If a longitudinal space defines an entire hull then SPACEN = 0.

As an example of the use of *longitudinal spaces* let us suppose that two variants of the same vessel hull are to be studied, one with a bulbous bow, the other without. The bulbless hull can then be described as longitudinal space 0 and the bulb separately as longitudinal space -1 (see the Appendix, relation **LSPACES** and the manual of the program ARCHIMEDES [1] for the convention for assigning space identifiers). The bulbless hull will be automatically considered in all hydrostatic calculations. The buoyancy of the bulb will be added only when its SPACEN value -1 will be appropriately included in the input.

Relation **COMPART** lists the identifying numbers of the transversal sections used to define a longitudinal space. This relation contains a tuple for each transversal section. The sections are generically called *stations.*

The longitudinal coordinates of the stations, measured from an origin defined in relation **LSPACES,** are stored in relation **STATNX.** Other attributes of the relation **STATNX** show if the transversal section is a *geometrical station* appearing in the theoretical lines drawing, a *frame* belonging to a steel plan and so on. Binary-valued attributes of this relation indicate for each station if it shall be automatically called by certain programs, for instance by a program for hydrostatic calculations, by a program for whipping calculations or by a program for seakeeping calculations. Thus, even if the sets of stations used in different computations may not be identical, all transversal sections are stored in one and the same relation. This procedure reduces data duplication in the frequent case in which the above mentioned sets overlap.

The stations are described in relation **OFFSETS,** which stores the height and the transversal coordinates of a number of points along the section.This relation contains a tuple for each point and its key is composed of the hull identifier, called HULLNO, of the station identifier, called STATNO and of the point identifier, called POINTN. This uniquely identifies each tuple. Further attributes show if the point belongs to a certain "longitudinal line", for example waterline 1, buttock 4 or knuckle 3. This kind of data is useful for graphic or CAD programs. The last attributes of this relation indicate which points shall be automatically employed in certain programs, for example for hydrostatic calculations or in seakeeping programs using the Frank close-fit method. This facility is important because many programs impose a limit on the number of points that may be processed and therefore these points must be conveniently chosen.

In damage calculations it is usually important to know if a certain line or some non-watertight openings are submerged. Thus, most regulations require that a *margin line* running 75 mm (3 in) under the bulkhead deck shall not be submerged when a number of compartments are flooded, and that unrestricted flooding due to immersion of openings shall be avoided. Examples of openings that shall not submerge under certain specified conditions are tank overflows and air intakes.

Checking the above requirements amounts to verify if at least one point of an imaginary "freeboard line" submerges. In order to make such checking calculations automatic, relation **FREEBRD** stores the three coordinates of the points that define the *freeboard line* and the openings that could cause unrestricted flooding.

The branch labeled APPENDAGES in Fig. 1 shows relations that store the size and location of appendages. Going from the top downwards we encounter the names of the relations that contain data on bilge keels, skegs, rudders, stabilizer fins, propeller shafts, shaft brackets and transom flaps. These relations may be used for calculating the performances of a planing hull with appendages (for example, by Hadeler's method [24], [25]), and for estimating damping factors in seakeeping programs.

Stability calculations requiring wind heeling-arm, and air- and wind-resistance calculations need data on longitudinal and transversal above-water ship outlines. The relation **CONDES** lists such contours by assigning unique contour numbers for a given SHIPNO value. A ship may have different contours for different load cases. For example, the lateral projection of a container ship is not the same with or without containers on deck. Consequently, several longitudinal and transverse contours may be defined for one ship and their relationship to the load case is shown in relation **LDCASES,** as discussed later on.

The relation **CONTOUR** describes longitudinal contours, that is the projections of ship above-deck outlines on a plane parallel to the center-line plane. Each contour consists of a succession of straight-line segments defined by the longitudinal and height coordinates of their end points. The under-deck contours need not be described here as they are part of the hull lines previously discussed. In a similar way, the relation **TCONT** defines transverse above-deck contours.

The "master" relation used in flooding calculations is **COMPNAM,** which lists the compartments of each ship, their identifying numbers, COMPNO, and compartment types (store room, tank).

A compartment is described as one solid, when its form corresponds to one of the five standard compartment types defined in the system. Otherwise it must be divided into several solids called *subspaces.* Each solid is delimited by a part of the hull surface and/or several planes. Relation **SUBSPCE** defines the geometry of the subspaces by describing the coordinates of the vertices of the delimiting planes. For the five types of subspaces that may be defined see the manual of the program ARCHIMEDES [1].

Relation **FLDCASE** lists the number of flooding cases to be checked for each ship. The combinations of flooded compartments in each flooding case is listed in the relation **FLDCOMP.**

Relation **FLDPERM** contains flooding library data. This relation assigns a volume permeability, VPERM, to a set of ordered pairs COMPTYP, LCASE. For a given ship, COMPTYP is the compartment type read from relation **COMPNAM** and LDTYPE the load case type retrieved from relation **LDCASES.** This relation is necessary because some compartments may have different permeabilities in different load cases. For example, the DDS 079-1 regulations of the US Navy specify a permeability equal to 0.8 for stores in the "full load" condition and 0.95 in the light operating condition.

MASS DATA

Mass data play crucial roles in naval architecture and they must be considered under several aspects. First, mass data can be allocated to different ship items or sets of ship items. Thus, each ship plate, ship stringer and ship engine can be assigned its own mass, center of gravity and moments of inertia. Furthermore, one can consider the mass, the center of gravity and the moments of inertia of sets of ship items such as the steel hull, the main machinery system or the whole ship. Under this aspect mass properties are *set functions.*

A formal definition of the concept of *set function* and its application to mass properties of ship items are given in [23]. What interests us is the *additive* property of set functions. For example, for sets of ship items that have no common elements *(disjoint sets),* the mass of the *union* of these sets is equal to the sum of the masses of the individual items. Thus, the mass of the steel hull is equal to the sum of the masses of its components. Other mass properties that are set functions and therefore enjoy the additive property are moments and moments of inertia.

In order to treat mass properties such as masses, centers of gravity, moments and moments of inertia as set functions and mechanize the use of their additive property it is necessary to classify the various ship items and organize them in a suitable structure. Ship items may be classified in accordance with criteria such as their function, the location within the ship or the manufacturing workshop. The best known classification criterion is the functional one. Examples of functional classification systems are the MARAD system of the U.S. Maritime Administration, the Ship Work Breakdown System (SWBS) of the U.S. Navy and the SFI group system of the Norwegian Shipping Research Institute. In designing the TECHNION DB we required that it should be applicable to any classification system chosen by the user.

Under another aspect ship data may be related to ship coordinates, in practice almost exclusively to longitudinal coordinates. For example, in bending moment calculations it is important to know the longitudinal distribution of the hull weights. The mass data stored in a DB shall be sufficient to generate such distributions.

The database is so designed that it fully takes into account the dual aspect of mass properties which are at the same time set functions and functions of coordinates. In order to achieve this, for each mass item the database clearly indicates its appartenance to a weight subgroup, to a load case (full load departure, ballast departure etc) if it is a deadweight item, and certain relevant coordinates.

Another problem with mass data is that different calculations employing them require different input data. Thus, in order to find the draft, the trim and the heel of a ship it is necessary to know the displacement and the three coordinates of its center of gravity. The same is true for intact and damage stability calculations. Programs that calculate ship motions require in addition some components of the ship tensor of inertia. Longitudinal bending moment calculations are based on the distribution of ship masses along the ship length. For the calculation of bending and torsional moments in waves it is necessary to lump the ship mass into mass sections. Then, the mass, the center of gravity and some components of the tensor of inertia of each mass section must be known. The mass data stored in the DB must be sufficient for generating all the above mentioned input sets.

The fourth problem dealt with in the TECHNION system is the avoidance of *derived data.* Simple examples of derived mass data are the traditional *weight summaries* (see, for instance, pp. 60-61 in [26] and pp. 32, 36, 40 in [27]). The sums of weights over weight subgroups and weight groups, the lightship displacement and the full load displacement are derived data and their presence in a database would be redundant as these totals may be calculated by simple addition whenever necessary. If one single weight item is updated, all partial and total sums must be updated too. Failing to do so would result in erroneous calculations.

Another example of derived data are the mass properties of bulk loads such as grains, or ores contained in holds or liquids stored in tanks. If the geometries of the holds and of the tanks are known, the masses, centers of gravity and moments of inertia of the bulk loads can be obtained by *capacity calculations.* Therefore, storing the mass properties of bulk loads means storing derived data. As generally true for redundant data, storing the mass properties of bulk loads creates a potential source of updating errors.

The four problems described above were analyzed in detail before proceeding to the design of the TECHNION database. The guidelines developed from this analysis may be summarized as follows:

1) Provide the possibility of storing mass data of items classified according to any system.

2) Describe the classification of ship items so as to enable automatic addition of mass properties over subgroups, groups and main groups of items.

3) Define a minimal set of mass data from which all input data required by known programs can be derived.

4) Design the relations containing mass data so as to reduce redundancy and especially avoid the storage of derived data. The main consequence of this requirement is that the mass data of bulk loads are stored in a way that completely differs from that in which discrete-load data are stored. Another consequence is that weight summaries are not stored at all.

The implementation of guidelines 1) and 2) above is based on the formal analysis described in [23]. The general idea may be summarized as follows. Ship items are identified in the database by *compound* identification numbers which are unique for a given ship. The first digits of this number constitute a *classification number* defined in accordance with some classification system (SWBS, MARAD, SFI etc). The last digits form a unique *identification number* within the subset defined by the classification number. In the original DB design the compound classification/identification number was six-digit long. Later experience lead to the augmentation of this number to seven digits. In some cases certain digits are replaced by alphabetic characters. We use the term *identifying number* for this case too.

In order to use the identification numbers for automatic summation of mass properties over subgroups, groups and main groups of ship items, the identifiers must reflect the tree (hierarchic) structure induced by the classification system. For example, in the SWBS classification system the main group 100 means the hull structure. All components of this structure have identification numbers beginning with 1. In order to obtain the hull mass, a computer program must sum the masses of all ship items the identifiers of which begin with 1. The analysis carried on in [23] showed that this is not always straightforward in known classification systems and it is clearly impossible in the MARAD system. Remedies are proposed in the quoted paper and applied in the system developed at the TECHNION.

Implementation of the third guideline required careful thinking. It was found that most input requirements of available programs may be implemented if the following data are known for all ship items:

item mass
longitudinal coordinate of item center of gravity
transverse coordinate of center of gravity
height coordinate of center of gravity
longitudinal coordinate of aftermost item point
longitudinal coordinate of foremost item point

The most advanced use of the above data is in calculating the *distribution functions* of mass properties introduced in [22].

The *mass distribution function* $W_i(x)$ of the $i-$ th ship item is defined such that the mass of this item contained between the transversal planes $x=a$ and $x=b$ is

$$W_i[a,b) = W_i(b) - W_i(a) \tag{1}$$

Distribution functions of the first and second moments of the same item can be derived from the mass distribution function by integration and/or multiplication.

Several uses of the distribution functions of mass properties are described below in the chapter on examples of applications. When employing distribution functions of mass properties it is possible to treat concentrated and distributed masses in practically the same way.

The data set described above is directly stored in the database for discrete load items such as all lightship items and non-bulk deadweight items. In accordance with the fourth guideline, the same data are not stored for bulk loads. For such a load item the database contains the identifier of the tank or hold that contains it, the degree to which this tank/hold is filled, and the density of the load. From these data it is possible to derive the same set that was described for discrete loads.

After the above general discussion we shall go now into details of the present implementation. The relation **WEIGSYS** defines for each ship the system used for classifying its items. The complete descriptions of the classification systems encountered in the database are stored in the library relation **CLASSYS.**

The mass data of lightship items are contained in relation **LIGHTWT.** The key of this relation is composed of the ship number, SHIPNO, and the compound identification number of the item. The other attributes are the mass, the coordinates of the center of gravity and the longitudinal extension of the respective item.

Deadweight items must be employed in calculations according to the load case involved, for instance full load departure, full load arrival, maximum operating condition. The various load cases are listed in relation **LDCASES,** which also associates to each case a longitudinal and a transversal contour. Another attribute stored in this relation is LDTYPE which is used to chose the permeability of flooded compartments as described above for relation **FLDPERM.**

The relation **DISCLD** stores the mass data of deadweight/payload items such as containers, vehicles and airplanes. The data set is the same as in relation LIGHTWT, plus the key attribute LCASE which relates each item to one of the load cases stored in relation LDCASES. It is this attribute which automates the selection of items to be processed for a given load case.

The bulk-load data set is stored in relation **BULKLD,** the key of which is similar to that of relation **DISCLD.** The non-key attributes are the compartment number, COMPNO, the filled capacity/total capacity ratio, FILLIN, and the density of the bulk load, DENSIT. Obviously, the mass of a bulk load can be obtained by multiplying the volume of the containing compartment by the values of the attributes FILLIN and DENSIT. The center of gravity is the center of the volume that contains this mass.

In the preliminary design stage it is not practical to produce all the data stored in relations **DISCLD** and **BULKLD.** Therefore, a special relation was designed for preliminary deadweight calculations. This is **DEADWGT** and it contains a restricted data set. The distinction between discrete and bulk loads is not made in this relation.

As we gathered experience with our system we became aware that further reduction of redundancy is possible. This is true for those items which repeat themselves several times on the same ship or are common to several ships. Consider, for instance, a specific valve. A ship can contain many valves of the same type and size. Repeating their mass for each one of their occurrences is a form of redundancy. It is possible to store the mass only once, in a separate relation **MASSMEL,** and list only the occurrences and positions of these valves in relation **MELLGWT.** Then relation **LIGHTWT** will contain only those items that do not repeat themselves. Similarly, the masses of repetitive discrete-load items can be stored in a relation called **MELDLD** and then the relation **DISCLD** will also contain only those items that do not repeat themselves.

Further developments can be contemplated. For example, instead of having one relation **MASSMEL,** to have one MEL relation for electric components, another one for hydraulic components and so on. These specific relations can store more attributes. Thus, possible details of the relation for electric components may be the rating in kW, the voltage, the degree of protection and so on.

STRUCTURAL DATA

Two relations are envisaged for the moment. At the ship-data level relation **SECPROP** will contain data necessary in the calculation of sectional moments of inertia. At the library-data level relation **MATERLS** stores the density, ρ, yield stress, S_y, ultimate stress, S_u, elongation at S_y, Poisson's ratio, μ, and the Young modulus, E, of structural materials.

PROPULSION

Propulsion data cover ship resistance, self-propulsion and propulsion hardware, that is engines and propellers.

The general model-test data are stored in relations **MODEL** and **RUNDATA.** The former has as key the model identifier, MODNO, and the variant identifier, MODVAR. An attribute called MODREF indicates if the tests were run with a model, described in relation **HULL,** or with a full-scale ship described in relation **SHIPS.** Further details are the name of the basin that carried on the tests, the test-basin number of the model and the characteristic length with which test results are nondimensionalized in relation **RESIST.** This latter one may be the ship length, for conventional displacement ships, the beam, for planing hulls, or the cubic root of the volume of displacement, $\nabla^{1/3}$, for vessels in the preliminary design stage.

Relation **RUNDATA** describes the test conditions: draft, trim, water kind (fresh or salt) and temperature and the correlation allowance assumed by the test basin.

The nondimensionalized test results are contained in relation **RESIST** which lists for each Froude number the coefficient of residual resistance, the dynamic trim in radians and the dynamic sinkage-to-characteristic-length ratio.

Relation **WETSURF** contains the nondimensionalized wetted surface. In general this data item is not redundant although it is computable from hull geometry. Indeed, it should include the wetted surface of appendages for ships for which the hull geometry is stored in the DB. Also, it shall be used for ships with unknown geometry when their wetted-surface values can be obtained from either test-basin publications or approximate formulae.

Relation **RAPPEND** stores the scaled-up resistance of appendages when such data are available.

Two library-data relations necessary in resistance calculations are mentioned at the beginning of this chapter. They are **KVISC,** which stores kinematic viscosity values, and **WATDENS,** which lists water-density values for fresh and salt water at a number of frequently encountered temperatures.

The relationship between ship, engine/s and propeller/s is described by the three relations **SELFPRO, PROPINS** and **ENGINST.** It is not always easy to define this relationship. There are many possible combinations, such as two engines on one shaft, two shafts for one engine or even more complicated arrangements, for example a CODOG propulsion plant in which two shafts are driven each one by a diesel engine at cruising speed and both by a gas turbine at high speeds.

Relation **PROPINS** defines for each shaft the propellers installed on it and the direction of rotation (right-handed or left-handed). The depth of the propeller axis below the water surface, necessary for the calculation of the cavitation condition, appears in relation **SHAFT,** which is one of the relations describing appendages. Relation **SELFPRO** lists for stored pairs {propeller, ship} the performance coefficients obtained from self-propulsion tests: wake fraction, thrust deduction factor and relative rotative efficiency.

Relation **ENGINST** lists for each mode of operation (cruise, top speed etc) and direction (ahead, astern) the installed engine, the gear ratio and the mechanical efficiency.

Data of propulsion hardware are naturally stored in MEL relations. Thus, relation **ENGINES** contains general engine data such as manufacturer's name, catalog type, kind (diesel engine, gas turbine, steam turbine), mass, overall physical dimensions, rated output and rpm.

Relation **DIESDAT** lists the power-limiting curves of diesel engines, as functions of rpm and operating conditions.

The specific fuel consumption of diesel engines as a function of operating condition, rpm and power is stored in relation **DIESSFC.**

For gas turbines relation **TURBDAT** describes the power-limiting curves as functions of the operating condition, rpm and the air-inlet temperature of the power turbine. The specific fuel consumption of gas turbines is stored in relation **TURBSFC.**

A number of standard combinations of operating conditions are described in relation **OPERCOND.** These data include air and water temperature and intake and exhaust pressures.

The other propulsion hardware components in the DB are propellers. We distinguish between propeller series for which nondimensional data are stored and actually built or designed propellers for which dimensional particulars are available.

The "master" MEL relation for propellers is **PROPSER.** It stores the general data of propeller series; the name of the propeller series or of the manufacturer, the identifier within the respective series, the number

of blades, the expanded-area ratio, a pitch characterization (fixed, controllable etc) and the operating regime (undercavitating, supercavitating). These data are nondimensonal.

The main dimensional particulars of propellers actually existing in an inventory or catalog are stored in relation **PROPPAR.** These data include the diameter, the pitch-to-diameter ratio, the sense of rotation, the hub diameter, the mass, and the moment of inertia.

The nondimensional open-water performances of propeller series are listed in relation **PROPDAT.** The key is composed of the propeller identifier, the pitch-to-diameter ratio, the cavitation number, σ, and the coefficient of advance, J. The non-key attributes are the thrust coefficient, K_T, and the torque coefficient, K_Q. The open-water efficiency, η_o, is not stored because it may be derived from the other stored data.

The geometric dimensional data of actual propellers are defined in relation **PROPGEO** which contains the pitch distribution and the blade thickness and chord.

Program sequences for complex calculations

In naval architecture it is often necessary to run a sequence of programs in order to obtain the desired results. Let us consider a relatively simple example in which the draft and the trim of a ship is to be calculated for a given loading condition. A possible flow chart of this computation is shown in Fig. 2. The sequence starts by calculation of the capacities of tanks and holds that contain bulk loads. From these results the user obtains the masses and the centers of gravity of the bulk loads. The masses and centers of gravity of the lightship and of the discrete loads are calculated separately and added to those of bulk loads. This yields the ship displacement, Δ, and the coordinates *LCG* and *VCG* of its center of gravity. These data together with a hull description are the input of a program for hydrostatic calculations that finds the desired draft and trim. The naval architect has therefore to employ a number of different programs, i.e. a program for capacity calculations, a program for weight calculations and a program for hydrostatic calculations that finds the floating condition by some iterative procedure. It is also the responsibility of the naval architect to transfer data between the programs, e.g. the weight of the bulk loads, as computed with the capacity program, to the program that computes the displacement. This manual handling of programs and data files by the naval architect is both cumbersome and a source of operating errors.

This section discusses methods for the automatic execution of several programs in a sequence, by one single computer command. This enables the naval architect to accomplish a complex sequence of computations without having to know which programs and files are involved. The use of one single command is a significant simplification of the operation, and it eliminates all the possibilities for erroneous handling of programs and data files.

How can a number of existing ship analysis programs be combined into one unit that executes a certain sequence of calculation? This problem is not particular to naval architecture. Weck and Carl [29], for example, discuss it in relation with general mechanical design. They distinguish three solutions. The first solution is to transform one program in a *main program* and the others in its *subroutines*. The second solution consists in transforming all programs in subroutines and writing a main program that calls them in the correct sequence. The third solution consists in keeping all programs as they are and calling them from a *command program* written in the command language of the operating system of the computer employed.

Transforming programs into subroutines involves rewriting some parts of them. This requires a thorough knowledge of the program and implies some work and much attention. Another problem is that of transfering data between the main program and its subroutines. The safest way is to transfer the data as *subroutine parameters*. However, if the number of parameters is large this method is not practical. In another solution, provided in FORTRAN, data is transferred to subroutines through *COMMON blocks*. It is necessary that all subroutines employ exactly the same COMMON structures. If it is required for some reason to change a COMMON block, all the corresponding blocks in other subroutines must be corrected in the same way. Failing to do so would yield errors.

We have seen that the integration of a number of existing FORTRAN programs into a single program that executes a required sequence of computations involves several problems and is in some cases not practical. We, therefore, preferred a third method based on storing in one file all the operating system commands that activate the required sequence of ship analysis programs. The operating system may thereafter get one single instruction that tells it to execute all the commands stored in this file. This single instruction is all what the naval architect has to give the system in order to get his sequence of computations done. This solution assumes that the programs in the sequence can get their input data in an automatic way. As previously explained, new ship analysis programs are designed to retrieve their data directly from the database, while old programs get their data from *intermediate files*. These files are produced by *interface programs*, which retrieve the required data from the database and convert them into the organization required by the particular programs. The commands that activate the interface programs are also stored in the command file.

An interesting programming problem regarded the identification of the different intermediate files, such that the different ship programs can get the right input data files. The solution was to use twin-component names for the intermediate files. The first component of the name identifies the program to which it is destined, while the other component characterizes the object described by the data.

The realization of twin-component file names with the VM/CMS operating system employed on TECHNION's IBM 3081 computer was straightforward. A file in this system is identified by three components:

file name, abbreviated to *Fn*, *file type, denoted Ft and file mode* abbreviated to *Fm*.

Our system constructs automatically the name of the file such that *Fn* represents the identifier of the object, for example a ship, described by the data retrieved from the database, whereas *Ft* identifies the program that employs the data in the file as its input. As an example let us assume that hydrostatic calculations are required for a hull numbered 18 in the database. The hull description in the database is in a standard form and must be converted to the input format of the program ARCHIMEDES, that is an input file called Block 4 [1]. In our system the interface program PBLOCK4 retrieves the data of hull No. 18 and produces a file with Fn = "S00018" and Ft = "BLOCK4". This automatically generated file name enables the ARCHIMEDES program to employ the right file of input data. Similarly, the output file produced by ARCHIMEDES is given the Ft value required by the program that employs this file as its input.

In the VMS, UNIX and DOS computer operating systems, the role of *Ft*, is played by the *file extension*. Thus, the name of the file exemplified above could become "S00018.BL4".

In conclusion, the techniques of storing all the required operating system commands in a single file enables a sequence of programs to be executed completely automatically. The data retrieval and data file naming is also done automatically by commands in this file. All what the naval architect has to do is to issue one single command, which is usually the name of the file that stores the commands. The use of files of operating-system commands as described above has a number of advantages, which are discussed below in some detail.

Programs of various sources may easily be integrated into one system. Existing, old programs are regarded as "black boxes" which receive some input files and produce some output files. It is thus not required to know anything about the internal workings of the old ship analysis programs, nor is it required to modify them in any way in order to integrate them into the system. This is especially useful when it is necessary to employ old proven programs, whose internal structure is poorly documented, and where the original programmer is no longer available. The black box approach is also applicable with most commercially available programs that are supplied in unreadable "object code" (the native language of the computer). The vendors of these programs are normally not interested in disclosing how their products work.

Any programming language may be employed for the construction of ship analysis programs. Since the programs are considered as black boxes the way in which they are implemented internally is immaterial. It is thus possible to combine purchased object code programs with in-house produced FORTRAN programs. With these techniques FORTRAN is no longer the only permitted language. The only requirement made is that the programs are designed to work with the common standard database. We believe that most programs will continue to be written in FORTRAN because of a number of well known reasons. However, if it is required to produce a program in short time, and if performance is not a problem, interpretative languages, such as BASIC, APL and REXX may be preferred. REXX is an example of a modern operating system command language. It is so powerful that it may also be used for computations (see, for example, [30]).

The file of commands enables the application of available software packages for statistics, graphics etc. An example was the use of SASGRAPH for plotting of data produced by one of our ship analysis programs. The graphs were one of the results which were achieved automatically by calling the file of commands.

The use of files of operating system commands has also a number of problems. The use of such operating system commands is somewhat expensive. It is typically measured in milliseconds of computer time, as compared to the micro seconds required to call a FORTRAN subroutine. However, since the number of operating system commands to be executed is usually small, the overhead involved is normally negligible. When we implemented our STABIL program for intact stability calculations, we used the ARCHIMEDES program for hydrostatic calculations as a FORTRAN subroutine to be called from STABIL. This solution is more economical than calling ARCHIMEDES through operating system commands. We wonder, however, today whether the savings were worth the effort.

Unfortunately different operating systems employ different command languages. The porting of a program to a computer employing a different operating system involves therefore the translation of the operating system commands into the language employed on the other computer. However, since the number of commands to be translated is usually very small, this is a relatively minor effort. Translation of operating system commands may be saved if the two computers employ the same language. An example is the REXX command language which works on both IBM mainframes and on IBM PC. Another important example is the UNIX operating system which works on many different computers from personal to supercomputers.

Examples of application

PRELIMINARY DESIGN

When we started the research at TECHNION our intention was to develop a database suitable for preliminary ship design. Our approach was motivated by the observation that preliminary studies are usually based on previous ship designs. This past experience is expressed in large amounts of data which constitute perhaps the most valuable asset of a ship designer. The problem was how to organize this database and its management system so as to allow for its most efficient use. Other researchers too have realized that an efficient method of dealing with these data is needed. Thus, Allieri and Sartori [13] write: *"The ship designing ... recourse to 'past history', that is to say to 'experience' in the creation of new projects is part of the normal practice, and the need for an instrument allowing information to be manipulated in an agile and efficient manner is particularly felt ...".*

We gradually discovered that the database can be used not only during all design stages, but also throughout the entire ship life. Therefore, eventually our database developed to contain also data beyond those required for preliminary design. Still, the use in preliminary ship design involves some of the most ingenious and creative applications in naval architecture. Therefore we shall begin with this stage.

The most favorable design situation is that in which the "new ship" is derived from a "parent ship". Several methods are known. The simplest is the *percentage method* proposed by Evers [31]. It is based on the assumption that the mass W_i of the i-th weight group represents the same percentage of the displacement in the parent ship and in the new ship. For example, the hull-weight-to-displacement ratio is the same in the old and in the new ship. Then, specifying the absolute change of one weight group, the new displacement is found by a very simple calculation.

In a *differential* method, such as that of Bubnov (see, for example, [32]), the required weight change is linearly related to the changes in the main dimensions. The coefficients in this expression are obtained from the masses of the parent ship.

In a third group of methods, based on less simplifying assumptions, it is possible to write the Archimedes principle as a nonlinear algebraic equation in which the unknown is either one of the main dimensions, usually the ship beam, B, [31], [33] and [34], or the ship displacement, Δ [35]. The coefficients of this equation can be derived from the masses of the parent ship and from the ratios of her dimensions.

The masses used in all the design methods mentioned above are those of "weight groups" defined such that it is possible to establish for each one of them a law of similitude (ratiocination). For example, under the simplest assumption the hull mass may be considered proportional to the *cubic numeral* $L \times B \times D$, and the mass of the main machinery proportional to the installed power.

In the TECHNION system the masses of the weight groups are obtained by summing the masses of the individual items, which are stored in the database. For the weight groups that correspond to the classification system employed for the parent ship, the summation is made by a program called WGHTOTAL which works as described in [23]. The designer may, however, wish to organize the masses into different groups, more suitable for preliminary ship design. This task is facilitated by the data selection tools provided in present-day database management systems.

If the design is not based on a parent ship, but the new ship is of a type, and falls within a size-range for which the database contains enough examples, the coefficients of the nonlinear design equation can be obtained by statistical methods. With data organized in normalized tables, as in the TECHNION database, it is relatively easy to perform the required regression analysis. The task is again made easier by using tools of the database management system, as will be exemplified in the coming paragraph.

A classical problem in preliminary ship design is the determination of ship "proportions", namely the length-to-breadth ratio, L/B, the breadth-to-draft ratio, B/T, and the coefficients of form C_B, C_P and so on. This problem may be solved as follows. Data of suitable ships or hulls can be *selected* from DB tables by the *SELECT* function of the relational database management system. This function can be invoked with a condition, for instance "select all ships the length of which is between 100 and 120 m and the speed between 16 and 18 knots." It remains to the user to write a program which produces a file with the L/B and B/T ratios of the selected ships, together with corresponding displacement data, and to feed this file to an available regression program. At the TECHNION we carried on all these operations in one experimental program and found this way of operation to be practical.

The database can also facilitate the design of ship lines. The database can store many hull forms in the relations **HULL, LSPACES, COMPART, STATNX** and **OFFSETS.** If one of the stored hull forms is suitable as it is, it can be selected and scaled to the required dimensions. If one or both ratios L/B, B/T must be changed, the program ARCHIMEDES provides an easy way to do this by using different scale factors on the three axes of coordinates. The hull obtained by this *affine transformation* conserves, however, all coefficients of form of the parent hull. Other transformations are necessary to modify the coefficients of form. This can be achieved, for instance, by multiplying the distances between stations by a factor which changes along the hull or by multiplying the half breadths by factors which change with the height above baseline and, possibly, also with the longitudinal position of the affected station (see, for instance, [36], [37]).

Transformations as described above can be readily performed when the data are stored in the database. Indeed, in the TECHNION DB the distances between stations are stored in relation **STATNX** and this is separate from relation **OFFSETS** which contains half-breadths. It is therefore relatively simple to use these data as variables in transformation routines. A routine for changing the distances between stations will employ only the relation **STATNX,** and a routine for changing half-breadths will use input from relation **OFFSETS.** If the transformation depends also on the longitudinal position of the station, an embedding loop can use data from relation **STATNX,** and the embedded loop, data from relation **OFFSETS.** We used such routines to distort bow stations and to check the effect of their flair on intact stability.

Once the lines have been chosen it is possible to evaluate the powering requirements of the new ship. If the basin-test results of the chosen hull are available in relation **RESIST,** resistance and powering calculations can be readily performed. For new hull forms approximate powering calculations can be performed using one of the known *ship series.* Alternatively, the designer may carry on his own statistics over data of a number of ships available in the DB. Statistics may also be employed in preliminary studies to obtain resistance performances as functions of significant parameters, such as coefficients of forms. With the aid of such statistics it is possible to find the "optimum" hull for a given task. A large database may constitute an excellent basis for studies as described above. Examples are presented in [38] and [39].

CALCULATING THE PARAMETERS OF THE FLOATING CONDITION

The parameters of the floating condition are calculated by the program FLTCOND. This program takes full advantage of the DB in the sense that, once the ship identifier, SHIPNO, and the load case identifier, LCASE, have been entered by the user, the program automatically retrieves the hull description and the mass properties of all items aboard. The only additional input may be the water density, if this is different from the

default value 1.025 t/m^3. The output consists of the mean, forward and aft drafts, heel, trim, metacentric height, waterline length and minimum freeboard.

The flow of data is shown in Fig. 3. A first interface program, PBLOCK4, retrieves the hull description from the DB and produces the intermediate file required by the program ARCHIMEDES. Another interface program, WGHTOTAL, directly retrieves the data of discrete weight items from the DB and receives similar data of bulk loads processed by the interface subroutine BULKLD. BULKLD uses geometric and density data stored in the DB in order to calculate mass data. The program WGHTOTAL produces an intermediate file with file type (see chapter "Program sequences" above) WGHTOTAL. The program FLTCOND reads from this file the displacement and its moments relative to the three planes of coordinates. The program FLTCOND calls the program ARCHIMEDES using as input the intermediate files describing the hull, the displacement and the coordinates of the ship center of gravity. By an iterative process the program ARCHIMEDES finds the floating condition, that is the position of equilibrium.

The program FLTCOND can be used either as a stand-alone program or in a longer sequence of calculations in which it delivers the parameters of the floating condition to another program.

The central role of the database is well apparent in this example. The hull description and the mass data are stored in the database. If the user wants, for instance, to check the effect of a new load, he has only to add that load and trigger the sequence. The sequence of calculations is carried on automatically. If the new load was entered correctly, there is no other possibility for user introduced mistakes and the designer does not deal with irrelevant computer technicalities.

INTACT STABILITY

Intact stability calculations are performed by the program STABIL. Its flow of data is shown in Fig. 3. The user input consists of the ship and load case identifiers, the selection of the stability regulations involved, for instance U.S. Navy regulations DDS 079-1, and the criterion against which stability must be checked, for example combined wind and waves. When investigating the stability during turning it is necessary to enter also the ship speed and the turning radius.

The output of the STABIL program is a table containing the righting, the heeling and the residual arms (that is, the difference between the righting and the heeling arms) at a number of heeling angles. Furthermore, the program provides an estimator of the degree of compliance with the chosen stability criterion, such as the ratio of the area under the righting arm to the area under the heeling arm.

The first part of the STABIL program calls the program FLTCOND and finds the parameters of the floating condition as described above. Next, the program ARCHIMEDES is run for the calculation of the cross-curves of stability, and the righting arms are obtained for the center of gravity computed by the program WGHTOTAL at the beginning of the run.

In order to calculate the wind moment, the above-deck lateral outline is read from relation **CONTOUR** and completed with the under-deck outline as stored in relations **OFFSETS** and **STATNX.** The *sail area* is the area enclosed between the outline and the waterline calculated by FLTCOND. Its area and centroid are yielded by one of the subroutines.

The corrections for free-surface effects are calculated by the program BULKLD.

At the moment of writing the program STABIL checks stability in seaway and in turning according to the German Federal Navy regulations BV 1033 and the U.S. Navy regulations DDS 079-1. Further criteria and regulations are planned for the future.

The program STABIL is an example of complex computation involving data of many kinds: a hull description, the masses of a given load case, a sail area and free-surface effects. All these data are automatically derived from the DB. The long sequence of calculations is again done in a way that is completely transparent to the user. All calculations are carried on with only one data version and this is necessarily up-to-date. Unlike to previous practice, in which computer and manual calculations were mixed together, there is now no possibility at all that the user can pick data from different stages of the design process and blend them in an erroneous sequence of calculations. Also, in contrast to previous procedures, there is no need for the naval architect to look for the data through several documents, more than often dispersed between colleagues and departments. Typically, the whole sequence takes less than a minute of user time on TECHNION's mainframe IBM 3081. The time employed by the computer is actually much shorter as the facility may serve 200-300 users at the same time.

DAMAGE STABILITY

Damage stability calculations are performed by the program DAMAGE. Its flow of data is also shown in Fig. 3.

As in the program STABIL, the user has to supply only three data items: the ship and load case identifiers and the choice of regulations against which the vessel is to be checked. All other data are automatically derived from DB relations.

The program DAMAGE has common subroutines with the program STABIL, for instance the one for calculating the wind moment, and also uses the program FLTCOND. The sequence starts by finding the floating condition of the intact ship. This is done in the same way as in the program STABIL. The number of flooding (damage) cases to be checked is read from relation **FLDCASE,** and the identifiers of the compartments considered damaged in each flooding case are retrieved from relation **FLDCOMP.** The permeability assigned to each compartment is automatically obtained from relation **FLDPERM,** as function of the compartment type (retrieved from relation **COMPNAM),** load-case type (read from relation **LDCASES)** and, for holds and tanks, the degree of filling (stored in relation **BULKLD).**

For each damage case the program DAMAGE calls the program ARCHIMEDES as a subroutine and finds the floating condition after flooding. The wind heeling arm and the free surface effects of unflooded tanks are calculated in the same manner as in the program STABIL.

The results of calculations are used to check if the adopted criteria are fulfilled. Examples of such criteria are a minimal value of the residual arm and the requirement that no point of the "freeboard line" shall submerge. As explained above, the freeboard line is defined in relation **FREEBRD.**

From the following example it can be appreciated how powerful the program DAMAGE is. For a given vessel, at one load case, 17 different flooding cases had to be checked. The program was run on a

MICROVAX II mini-computer. The whole sequence took 10 minutes. Before the present version of the program DAMAGE was written, the identifiers of flooded compartments and the permeabilities had to be entered manually for each case. Running the same 17 cases was a matter of several hours of manual work. The impressive time savings are obtained by storing in the DB the definitions of flooding cases and enough data for automatic selection of permeabilities. Furthermore, once the data of a compartment have been stored, the conpartment can be used in any flooding combination without any additional work.

The programs FLTCOND, STABIL and DAMAGE can be operated also in a "manual mode". Then, only the hull description and the internal subdivision are retrieved from the DB, and the user has to enter the displacement, the coordinates of the center of gravity, the identifiers of flooded compartments and so on. This mode can be used for certain investigations in which some parameters do not yet have values stored in the DB. Computation times are obviously much longer.

SHIP MOTIONS

Programs for calculating ship motions, for example MIT's 5D program [40] and various variants of the SCORES program [41] include in their input:

a) the displacement and the coordinates of the center of gravity;

b) *geometric sectional properties,* such as the area, the vertical coordinate of the center of gravity and the girth of submerged transverse sections, if conformal mappings are used in calculations;

c) the coordinates of a number of points describing the submerged transverse sections, if the Frank close-fit method is used;

d) two or three radii of inertia related to the ship tensor of inertia;

e) wave-data describing the sea in which seakeeping performances are to be obtained.

The input described under a) and b) is derived from primary data permanently stored in the DB, and the input items listed under c) and e) are primary data permanently stored in the DB. The data included in d) are usually obtained from empiric formulae. A method for calculating the components of the tensor of inertia from DB data, for any load case, is described in [22].

At the TECHNION we wrote programs that prepare the input data of seakeeping programs and run the programs. The flow of data for the program SCORES is shown in Fig. 3. Once the ship and load case identifiers have been entered by the user, the displacement and the coordinates of the center of gravity are calculated by the programs WGHTOTAL and BULKLD. The floating condition is found by the program FLTCOND which calls the program ARCHIMEDES as a subroutine. The hydrostatics option of the program ARCHIMEDES is called with the draft, trim and heel yielded by FLTCOND. The results are the sectional properties which are fed to the program SCORES. Wave data can be retrieved from the database. Other parameters, such as ship speeds and headings relative to waves, as well as a specification of points at which accelerations are to be calculated, are entered interactively by the user.

With the interface programs designed at the TECHNION the input to the seakeeping programs can be prepared in a matter of minutes. This is particularly important when performances must be checked for several load cases. We recall cases when the designers spent much time in discussions about choosing a few, more critical, load cases, rather than going through the tedious job of preparing the input for all possible load cases.

BENDING MOMENT

In all applications discussed up to now the ship mass is considered concentrated at the center of gravity of the ship. In contrast with this, when calculating the longitudinal bending moment it is necessary to distribute the ship mass along the ship length. In traditional practice, the data needed for this were stored in another place and form than the data used to find the center of gravity of the ship. In the system developed at the TECHNION the same data are used to produce the concentrated mass model and the distributed mass model. The flow of data is schematically shown in Fig. 4 and the general algorithm, based on the theory developed in [22], is the following.

Data of discrete lightship and deadweight mass items are directly retrieved by the program WGHDSTR from the DB [22]. The data of each item includes the mass, DSMASS, the longitudinal coordinate of the center of gravity, XCG, the longitudinal coordinate of the aftermost item point, XAFT, and the longitudinal coordinate of the foremost item point, XFWD. If XAFT = XCG = XFWD then the mass is concentrated at the point defined by these three identical coordinates. Such a mass is described by a *step* function that is equal to zero aft of x = XCG, and equal to DSMASS from XCG forward. In the system of coordinates used to describe ship properties, x is the longitudinal coordinate. If XAFT $\neq$ XFWD the mass is distributed longitudinally. Then, if the center of gravity defined by XCG is situated between ⅓ and ⅔ of the distance XFWD - XAFT, the mass distribution corresponds to the traditional *trapezoidal mass density distribution.* In the system developed at the TECHNION, however, the mass distribution, rather than the mass density distribution is considered. This mass distribution is described by a function of x the value of which is zero aft of x = XAFT, it is represented by a second degree function between x > XAFT and x = XFWD, and equal to DSMASS forward of x = XFWD. When the center of gravity lies in either the first or the last third of the distance XFWD - XAFT, the item mass is decomposed in a concentrated and a distributed mass, such that the center of gravity of their resultant is identical to the given center of gravity. Each component is then treated as shown above.

The program BULKLD retrieves from the relation **BULKLD** the data of bulk loads and produces data equivalent to DSMASS, XCG, XAFT and XFWD. These data are further processed by the program WGHDSTR to yield the mass distribution functions described above.

The program WGHDSTR adds all the *mass distribution functions* pertinent to the load case under investigation and produces a *ship mass distribution function,* $W_s(x)$, which represents the mass of the ship comprised between its aft end and the transversal plane situated at any x , as defined by equation (1) in the chapter *The database.* The program WGHDSTR also produces a *ship mass moment distribution function,* $M_{yzs}(x)$, which expresses the moment, relative to a transverse reference plane, of the mass comprised between the same limits. The distribution functions $W_s(x)$ and $M_{yzs}(x)$ are used to produce the input to the program ARCHIMEDES which calculates the longitudinal bending moment.

The bending moment option of the program ARCHIMEDES accepts as input concentrated masses and distributed masses, the latter with a trapezoidal mass density distribution. The number of the masses is limited by the dimensions of the arrays that store the data. It may be said that any program for bending moment calculations written in FORTRAN suffers from similar limitations. The extent of the mass data of a given ship may exceed the dimensions of the defined arrays. This is certainly so for detailed naval ship designs in which the number of mass items may reach several thousands. The program WGHDSTR provides an option for reducing all ship masses to a required number of "trapezoids." Let this number be n. Then, the overall ship length is divided into n equal segments, the $i-$ th segment being delimited by its aft coordinate, x_i, and its forward coordinate, x_{i+1}. The mass of the $i-$ th segment is equal to

$$W_i = W_s(x_{i+1}) - W_s(x_i) \tag{2}$$

and the coordinate of its center of gravity is given by

$$XCG_i = \frac{M_{yzs}(x_{i+1}) - M_{yzs}(x_i)}{W_i} \tag{3}$$

The quintuples $\{ i, W_i, x_i, x_{i+1}, XCG_i \}$ are part of the input to the program ARCHIMEDES. The other part of the input is the description of the hull surface processed by the interface program PBLOCK4.

In a planned future version, structural properties will be retrieved from relation **SECPROP** and used to calculate primary deck and bottom stresses.

DYNAMIC SHIP LOADS

The programs for seakeeping calculations mentioned under SHIP MOTIONS (see [40] and [41]) include an option for the calculation of dynamic bending and torsional moments in waves. This option is based on a lumped-mass ship model. That is, the ship is divided along her length into a number of mass sections. For each section its mass, center of gravity and some radii of inertia must be supplied. A general method for doing this is described in [22]. It is based on the distribution functions of mass properties used also for bending moment calculations.

The flow of data in dynamic ship load calculations is shown in Fig. 4. It is the same as in the calculations of ship motions, except for the input of mass properties. For this input, instead of the mass, center of gravity and radii of inertia of the whole ship (supplied by the program WGHTOTAL), the program SCORES receives the same data set for each mass section in part. These sets are produced by the program WGHDSTR which uses primary data stored in the DB and data derived by the program BULKLD from primary data stored in the DB.

Assuming that the $i-$ th mass section is situated between x_i and x_{i+1}, its mass is given by equation (2); the longitudinal coordinate of its center of gravity, by equation (3). Other distribution functions introduced in [22] yield the other two coordinates of the center of gravity, a correct value of the moment of inertia relative to a transverse coordinate plane, yOz, and "optimistic" estimates of the other moments and of the products of inertia.

JUMBOISING

One way of upgrading a ship consists in cutting her somewhere in the parallel middle body and inserting there a supplementary section. This method is known as *jumboising* or *cut and splice*. When carrying on such a project it is necessary to evaluate the masses and the centers of gravity of the ship section aft of the cut plane, of the inserted section and of the section forward of the cut plane. The method of distribution functions of mass properties introduced in [22] for solving the problem of lumping the ship mass (see sections BENDING MOMENT and SHIP MOTIONS above) gives a simple possibility of performing jumboising calculations.

The advantage of the method of distribution functions is that, once the mass data are permanently stored in the DB, they are readily applicable for any cut position. Let us assume that the cut plane is situated at the longitudinal coordinate X_C. Then, the mass of the aft ship section is equal to

$$W_s(X_C)$$

and the mass of the fore section is given by

$$W_s(X_F) - W_s(X_C)$$

where X_F is the longitudinal coordinate of the foremost ship point. Expressions for the centers of gravity can be easily written and a possibility for estimating also the properties of the inserted section are shown in [22]. The flow of data is schematically represented in Fig. 4.

WHIPPING

The system developed at the TECHNION includes a program for calculating whipping stresses due to slamming in waves. This program was written by Shacham [28] and subsequently connected to the system described in this paper. The input includes a mass distribution produced by the program WGHDSTR, ship structural data (moments of inertia and position of neutral axes) directly retrieved from the DB and ship geometry also obtained from DB data. At the time of writing wave spectra are entered manually but it is planned for the future to retrieve them directly from the DB. The flow of data is shown in Fig. 4.

POWERING

A suite of programs for resistance, propulsion and powering is under development. The projected data flow is shown in Fig. 5. Two modes of operation are envisaged, one for ships the data of which are stored in the DB, the other for feasibility studies.

In the first mode of operation the user has to enter the ship and load case identifiers. Mass data are retrieved and processed by the programs WGHTOTAL and BULKLD. The floating condition is obtained by the program FLTCOND. Ship resistance data are retrieved from relation **RESIST** and processed by a program for resistance calculations. The resistance corresponding to the calculated draft is obtained by interpolation. Air resistance can be calculated using a frontal area based on data from relation **TCONT.** Added resistance in waves is obtained by means of one of the seakeeping programs. Such programs use DB data as shown under the title SHIP MOTIONS.

The engines and the propellers involved are read from relations **ENGINST, PROPINS** and **SELFPRO.** Diesel engine performances are retrieved from relation **DIESDAT;** gas turbine performances, from relation **TURBDAT,** and propeller performances, from **PROPDAT.** The effect of appendages is appreciated by using data from the relevant appendage relation.

In the option for early feasibility studies, hull, engine and propeller identifiers are manually entered by the user and then their data are automatically retrieved from the DB. Thus, the designer may, for instance, experiment with different propeller types.

Conclusions

The experience with the system described in this paper will now be summarized, and the methods enabling the performance of comprehensive computations with a high probability of correctness will be analyzed. The simplicity and power of the system are shown to be due primarily to the use of a single data standard. The paper will conclude with a discussion of the feasibility, benefits and problems of having a similar ship data standard for the entire industry.

The experience gained

Overall experience with the system proved very satisfactory. The automatic retrieval of data from the database has reduced the time required for ship analysis by an order of magnitude. The simplification of the retrieval process thus leaves more time for naval architects to use their engineering expertise in analyzing the results of calculations. With the old methods designers were sometimes so exhausted by the manual preparation of large amounts of input data that they limited their computer studies to a small number of cases. The system also enables the ship designer to check ideas instantly. In order to invoke a comprehensive computation the user needs to enter only a few key parameters. The rest of the data are automatically retrieved from the database. Checking a new ship configuration for stability, longitudinal strength, powering and seakeeping takes only a few minutes of user time. The engineer may thus accomplish a number of iterations in a short time. This is possible on current mainframes, which finish these kinds of computations in seconds.

The only cumbersome activity in our system is the entering of data into the computer database. Even data that have already been used in other computers must be entered manually into our system. If there existed a standard like the one proposed later in this section, for storing of ship data in computers, this manual, error-prone work could then be avoided. Neverteless, the entering of data into our system is easier in two respects, compared to many other systems. First, the data need be entered only once, since all the programs employ the same single database. This is in contrast to previous practice, in which every program has its own database and a configuration database also has to be maintained. Secondly, data correctness is checked to some extent against consistency rules. The checking is done at the time of data entry. Although these rules can only catch some gross errors, their usage is a help.

It proved practical to have a number of engineering catalogs (MEL relations) in our database, for storing performance characteristics of engines and propellers. Shipyards will likely extend this part of the database to all kinds of equipment. MEL relations may contain information on the consumption of electricity, fluids and air. This will enable the computing not only of weight, but also of electrical and other balances.

Reliability and consistency of complex studies

Because complex problems, such as the stability of a ship in damaged condition, are affected by a number of factors that influence the results in different directions, it is difficult for the naval architect to judge if the computer results make sense. The designer is completely dependent on the correctness of the programs and the data employed. The development of methods to ensure data and program consistency is therefore of crucial importance. The system developed at the TECHNION addresses data consistency-problems in a systematic way by employing the array of techniques summarized below:

* The same single database is employed by all programs. The database is normalized such that every piece of information appears only once. This guarantees that programs always employ the same, latest version of the data. An item in the database, moreover, needs to be updated only in one place, thus saving labor and reducing the possibilities of human errors. The same single database is employed for both configuration control and ship analysis. Discrepancies between configuration and design data are thus avoided.
* The database stores only primary data. Data that may be derived from these primary data are denoted "derived data." An example is displacement, which is derived from the weights of its components. Computing the derived data every time they are needed will ensure that they correspond to the latest ship configuration. It is felt that with the current, fast computers, the repeated computing of the derived data is an acceptable cost.

* The correctness of the data is checked against "consistency rules" at the moment the data are entered into the database.
* A single command can execute a complex sequence of calculations. The calling of the programs and the retrieval of the data from the database are automatically done, again saving labor and eliminating possibilities of human errors. The user may even not know which programs and databases are employed. A simple, straightforward method is employed to implement the single commands that execute the complex sequences of computations. These single commands carry out instructions that have been stored in a special file. The instructions call ship analysis programs and the interface programs that retrieve the required data from the database. These programs also convert the data to the special formats of the different programs.
* Old ship analysis programs, whose correctness has been demonstrated in numerous cases, may be readily employed in the system. This is done using the method described for sequences of calculations. The interface program to be constructed will produce an intermediate file, where the data are stored in the way expected by the old program.
* All database tables are manipulated by the same set of tools. One example is the single data-entry procedure employed for all the tables. Similarly, a single graphics program may be used for displaying the values in one table column, as a function of the values appearing in another column of the same table . This single graphics program works automatically for all database tables because it retrieves the specifications of the table column from the data dictionary of the system. The program will use this information to put the right scales and text on the coordinate axis. The user, therefore, must learn the use of only a single graphics program that works automatically for all the tables.

It is proposed that the same database be employed for both configuration control and ship design. The database, therefore, will contain more data than are strictly necessary for ship analysis purposes. This will ensure, however, that the results of the analysis correspond exactly to the current configuration of the ship. This assurance, it is felt, is worth the effort. Keeping a single database for both configuration control and design involves less effort than maintaining two separate databases for these two purposes.

A data standard for the ship industry.

All the components of the system -- i.e., database tables, general purpose tools and ship analysis programs -- were adapted to a single data standard. The specification of this standard is stored in the data dictionary, in the computer . Programs employ this stored specification in order to achieve a high degree of automation. If a standard, like the one used by our system, is adopted by the entire ship industry, further advantages may be gained. Such a standard will facilitate the exchange of databases and compatible ship-analysis programs between different organizations. Different companies working on the same project would be able to exchange data that are precisely defined in the standard, and in a format that is readily applicable to their computers. Manufacturers of engines, propellers and other equipment may supply their data in the same way.

The ship data standard will specify a normalized relational ship database schema similar to that shown in the appendix. This schema defines the tables composing a standardized ship database. Each of these tables will be stored in a separate file, in which the data will appear in the FORTRAN format specified for each column in the standard. FORTRAN programs, therefore, may read these files directly, without the need for any special database management system. Programs employing only input data files that meet the ship data standard may be said to satisfy the standard. Programs and files that meet the standard may readily be employed with each other.

For large ship CAD systems, it is an advantage to employ a commercial database management system (DBMS). These systems provide a number of facilities, including high-level database query languages. One of the relational languages, called SQL [42], became an ANSI standard in 1986. A FORTRAN program that has to retrieve or store data in a relational database may do so conveniently with a few embedded SQL statements. Such a program may in principle be ported from one DBMS to another. It should therefore be possible, for instance, to develop a system on a personal computer and later move it to a larger mainframe system when more performance is needed. If the SQL standard is to be widely accepted by the industry, it should also be supported by a ship data standard.

Relational databases are not optimal for all purposes. Hierarchical data structures, for instance, are more efficient for producing 3-D graphical representations. In [14], therefore, it is proposed to combine a hierarchical and a relational database for a ship product model. Computer scientists are also investigating data systems that combine relations with other data objects. At this stage, however, it will be useful to have a ship data standard based on relations. If hierarchical structures are standardized in the future, bridges between these two kinds of systems may then be defined. The usefulness of relations for storing ship data has been demonstrated in a number of computer systems, including the one described in this paper. This result is not surprising, since ship-data tables have been used successfully for generations. Another important advantage of normalized relational databases is that they provide a general purpose representation of the data, enabling the valuable information stored to be employed for many different purposes, including those not foreseen at the time that the database was established. Still another important reason for employing relational technology is that it has sound theoretical and practical foundations.

Information is one of the most important resources of the ship industry. In a highly competitive world, the ability to utilize this resource efficiently for many different purposes can be of crucial importance. As this paper has demonstrated, a relational database of ship engineering objects enables the naval architect to do many different kinds of studies. Furthermore, the principles and techniques behind such a database allow complex computations to be performed with a low probability of error. These techniques also lead to a high degree of computing automation, so that the naval architect can better concentrate on ship design issues. Finally, standardization of the database schema may enable a better utilization of available databases and programs. The exchange of data in a standardized format may also bring about more efficient coooperation among ship designers, builders and operators.

Acknowledgments

This research has been made possible through the combined efforts of many people to whom the authors express their grateful appreciation. The beginning was based on the programs obtained by courtesy of Prof. Heinrich Söding, then at the University of Hannover, and on the help of Prof. Abkowitz, from MIT. We especially wish to thank Mrs. P. Weisman, Mrs. M. Kalish, Mr. O. Immanuel, Dr. E. Schatz and Messrs. S. Lipiner, D. Livneh and M. Kossoy who participated in the development and implementation of the system. Of essential importance for the accomplishment of the research were the advice and support of Dr. I. Shacham, Mr. H. Genzer, Mr. D. Shacham, Dr. M. Naveh and Messrs. A. Naveh, Y. Shapir and R. Simon.

References

1. I. Poulsen, "User's Guide for the Program ARCHIMEDES 76 for Hydrostatic Calculations," translated and extended by A. Birbănescu-Biran, Haifa, Technion, 1982.

2. J. A. Belda, "Ship Design and Production by CAD/CAM," The Naval Architect, July, 1982.

3. J. Ikonen, "The integration of CAD/CAM Systems at Wärtsillä Shipyards," Computer Applications in the Automation of Shipyard Operation and Ship Design- ICCAS 85, Trieste, September, 1985, North-Holland.

4. N. Doelling, "Computer-Aided Preliminary Design of Ships," MIT Marine Industry Collegium Report 78-13, revised edition, 1978.

5. J. B. Woodward, editor, "The Computer System SPIRAL - Documentation for the User and the Module Writer," The University of Michigan - Dept. of Naval Architecture and Marine Engineering Report 212, 1979.

6. Jeffrey D. Ullman, Principles of Data Base Systems, Rockville - Maryland, Computer Science Press, 1982.

7. C. Delobel and M. Adiba, Bases de Doneées et Systèmes Relationnels, Paris, Dunod Informatique, 1982.

8. Günther Vinek, Paul Friedrick Rennert and A. Min Tjoa, Datenmodellierung: Theorie und Praxis des Datenbankentwurfs, Würzburg-Wien, Physica-Verlag, 1982.

9. François Fargette, Données de Base pour Bases de Données, Paris, Eyrolles, 1985.

10. C. J. Date, An Introduction to Database Systems, fourth edition, Reading-Massachusetts, Addison-Wesley, 1986.

11. Henry F. Korth and Abraham Silberschatz, Database System Concepts, McGraw-Hill, 1986.

12. Daozhong Xia, "An Approach of Integrated DBMS for CAD/CAM and MIS," Computer Applications in the Automation of Shipyard Operation and Ship Design V- ICCAS- 85, Trieste, September, 1985, North-Holland.

13. E. Allieri and G. A. Sartori, "Modulo Story: an Integrated System for the Storage, Retrieval and Handling of Ship Design Information based on a RDBMS," 25th CETENA Anniversary International Symposium, on Advanced Research for Ships and Shipping in the Nineties, S. Margherita Ligure - Italy, October, 1987.

14. Daniel W. Billinsley and J. Christopher Ryan, "A Computer System Architecture for Naval Ship Design, Construction, and Service Life Support," Trans. SNAME, 1986.

15. E. Kantorowitz, "Data Base and Program Organization Proposal," Danish Ship Research Laboratory, October, 1979.

16. E. Kantorowitz and A. Biran, "Computer Aided Preliminary Ship Design (CAPSD)," The 17th Israel Conference on Mechanical Engineering, Tel-Aviv, July, 1983.

17. Adrian Birbănescu-Biran, Ofer Imanuel, Marina Kalish *et alias*, "Computer-Aided Ship Design at the Technion - A Progress Report," Technion-Haifa, 1985.

18. E. Kantorowitz and Birbănescu-Biran, "A Powerful Integrated Ship Design System," The 20th Israel Conference on Mechanical Engineering, Tel-Aviv, June, 1986.

19. A. Biran and E. Kantorowitz, "Ship Design System Integrated Around a Relational Data Base," CADMO 86, International Conference on Computer Aided Design, Manufacture and Operation in Marine and Offshore Industries,Washington

20. A. Biran, J. Yanai and E. Kantorowitz, "Experience with the Application of a Relational Data Base to Naval Architecture," 21st Israel Conference on Mechanical Engineering, Haifa, June, 1987.

21. A. Biran, E. Kantorowitz and J. Yanai, "A Relational Data Base for Naval Architecture," 25th CETENA Anniversary International Symposium on Advanced Research for Ships and Shipping in the Nineties, S. Margherita Ligure - Italy, October 1987.

22. A Birbănescu-Biran, "Distribution Functions of Mass Properties - A Tool for Naval Architecture," Journal of Ship Research, December, 1987.

23. A Birbănescu-Biran, "Classification Systems for Ship Items - A Formal Approach and its Applications," Marine Technology and SNAME News, January, 1988.

24. J. B. Hadeler, "The Prediction of Power Performance on Planning Craft," Trans. SNAME, 1966.

25. Lawrence J. Doctors, "Hydrodynamics of High-Speed Small Craft," University of Michigan, Dept. of Naval Architecture and Marine Engineering No. 292, January, 1985.

26. C. S. Moore, "Intact Stability," in Principles of Naval Architecture, revised edition, edited by J. P. Comstock, New York, SNAME, 1977.

27. Ronald K. Kiss, "Mission Analysis and Basic Design,"in Ship Design and Construction, edited by Robert Tagart, New York, SNAME, 1980.

28. Itzhak Shacham, "Transient Response of a Ship Hull due to Wave Impact," Technion doctoral research thesis, Haifa, February, 1984.

29. Weck, M. and Carl, B., "Kopplung bestehender CAD-Programme zur heterogenen CAD-Systemen," KEM (Konstruktion, Elemente, Methoden), Nos. 10 and 12, 1984.

30. M. F. Cowlishaw, "The Design of the REXX Language," IBM Systems Journal, Vol. 23, N0. 4, 1984.

31. Wilhelm Hadeler, Kriegsschiffbau, Darmstadt, Wehr und Wissen Verlagsgesellschaft, 1968.

32. V. Laptĕv, Proiektirovaniea Morskih Kommerčeskih Sudov, 1st part, Leningrad, Gostransizdat, 1932.

33. A. Krause and E. Danckwardt, "Entwerfen von seegehenden Frachtschiffen und Fahrgastschiffen," Schffbautechnisches Handbuch, Vol. 2, edited by W. Henschke, Berlin, VEB Verlag Technik, 1964.

34. H. J. Westers, "Ontwerpen van Schepen 1," Technische Hogeschool Delft Collegedictaat, Delft, 1977.

35. George C. Manning, The Theory and Technique of Ship Design, New York, Technology Press of M.I.T. and John Wiley and Sons, 1956.

36. H. Flecken and K. Vogt, "Das Rechnerische verzerren von Schiffslinien", Hansa, November, 1971 and No. 11, 1972.

37. A. Koops, "Hull Form Definition and Computer Aided Design", Computer Applications in the Automation of Shipyard Operation and Ship Design- ICCAS85, Trieste, September, 1985

38. M. Carlier and M. O'Dogherty, "The Use of a Data Base in the Optimization Process of a New Design," ISSHES-83- International Symposium on Ship Hydrodynamics and Energy Saving, El Pardo, September, 1983.

39. Luigi Grossi, Claudio Camporese and Domenico de Stefano, " Utilization of Experimental Data Banks for Preliminary Ship design," La Marina Italiana, September-October, 1985.

40. Atle Steen, "User's Manual for the 5 Degrees of Freedom Seakeeping Program," Design Laboratory, Dept. of Ocean Engineering, Massachusetts Institute of Technology, Cambridge-Mass.

41. "User's Manual for SCORES II Program," Hydromechanics, Inc., Report No. 79-02, Plainview - N.Y., June, 1979.

42. C. J. Date, A Guide to the SQL Standard, Reading-Massachusetts, Addison-Wesley, 1987.

Appendix

Proposal for a ship-data standard

This appendix contains the skeleton of a proposal for a standard for ship databases. Because of space limitations only part of the relations to be defined in such a standard are shown here. A structured overview of a major part of the relations to be defined in the standard is shown in Fig. 1. Most of the relations specified here were employed in TECHNION's system. The specifications of this proposal were therefore influenced to some extent by the programs available at the TECHNION.

The purpose of the standard is to facilitate exchange of data and computer programs between different organizations. The standard is also intended to enable a number of different ship-analysis programs and ship databases to be readily integrated into one single system. The purpose of such an integration is to automatize to a large extent complex ship studies involving several programs and databases.

Programs that accept some of the database relations as their input may be readily exchanged between owners of standard ship databases.

The proposed ship database is composed of a number of tables *(relations)* the structures of which are defined in this appendix. These tables are designed as parts of a relational database. The tables may, therefore, be stored in, and be processed by any of the commercially available relational database management systems.

The ship database may be stored in either of the two forms described below.

1) As FORTRAN 77 files, that is every relation is stored in a separate file. The structure of the records *(tuples)* is defined in this appendix for each table in part.

2) In a relational database system. This system shall support the SQL standard query language.

For exchange of ship databases between different organizations only the first method, that is FORTRAN files, is employed. Both storage methods may be used for data processing by ship analysis programs. Ship analysis programs may, therefore, appear in two different versions:

- Programs that work directly with input data stored in FORTRAN files. These programs must be completed with all the software needed for retrieving the data from these files. The input are the files containing the tables described in this appendix.

- Programs that employ a relational database management system. Such programs must retrieve their input from the database through queries formulated in the SQL language. Therefore, only database systems that support the SQL standard may be employed. The purpose is to enable portability of the ship analysis programs.

The order of presentation of the tables *(relations)* in this appendix is that in which the various relations are met when traversing the functional tree of Fig. 1 from left to right, and each branch from top to bottom.

The principal description of each relation is enclosed in a box and consists of three lines:

line 1 - attribute (column) names. The names of the attributes that compose the identyfying key are underscored;

line 2 - Fortran 77 formats;

line 3 - domains. These are the sets from which the attribute values are taken. Examples of *domains* defined in this proposal are IDENTIFIER, NONDIM, LENGTH, MASS, TEXT.

An IDENTIFIER is a character string or an integer on which no arithmetic operations can be performed. Several identifiers can be *concatenated* to form a more meaningful identifier.

NONDIM is the domain of nondimensional numbers, that is numerical values without physical dimensions.

LENGTH and MASS have the usual physical meaning. They can be operands to arithmetic and comparison operations, subject to certain restrictions. For instance, a mass can be multiplied or divided by some power of a length, but not added or compared to any length.

A TEXT cannot be operand to any operator and neither can it be concatenated to any other character string.

The data dictionary of the database stores appropriate units and conversion factors for domains which have physical dimensions. SI units are prefered whenever possible. Exceptions can occur in a few cases in which other units are more convenient.

The length of attribute names is limited to six characters. Thus, these names may be readily used as variables in standard FORTRAN 77 programs, or in programs written in other languages in which the same length restriction is imposed.

Within each relation the tuples are stored in ascending lexicographic order of their key. This facilitates search procedures. It is therefore required that:

a) character-string values be left justified;

b) integer values be right justified.

The database system can supply certain default values, if the user does not fill in the corresponding values at data-entry time. The description of relations that follows contains a few examples of such default values. A special case of default values is that of *unknown* values.

A useful feature of database management systems is the ability of checking, to a certain extent, the correctness of data at entry time. For instance, the system shall not permit to enter a ship beam larger than the ship length. This is in contrast to old practice in which data correctness is checked at run time, that is data can be rejected when running the program or, still worse, errors are detected only after checking computation results.

In order to check data at entry time, *integrity constraints* must be defined in the system and enforced by it. The description of relations that follows shows a number of such constraints.

Throughout the following specifications it is assumed that x is the longitudinal coordinate, positive forwards, y, the transverse coordinate, positive to starboard, and z, the height coordinate, positive upwards. The origin of coordinates lies on the line of intersection between the centerline plane and the baseline plane. The longitudinal position of the origin may vary as shown in the appropriate relations.

GENERAL SHIP DATA

RELATION **SHIPS**

Application: This relation describes the main characteristics of ships. It actually provides a ship list analogous to Jane's books, or to the "Ships on Order" list published periodically by Motor Ship. The data contained in this relation may be used for preliminary design purposes. It is also used to establish scales in the input to hydrostatic calculations, for example in the intermediate file BLOCK4 of the program ARCHIMEDES.

SHIPNO	SHNAME	FLAG	SHTYPE
I05	A15	A04	A04
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

LS	BS	TS	DS	HULLNO
F07.03	F06.03	F06.03	F06.03	I05
LENGTH	LENGTH	LENGTH	LENGTH	IDENTIFIER

DISPLN	SPEEDM	SPEEDC	RANGE
F08.01	F04.01	F04.01	F04.01
DISPLACEMENT	SPEED_NAUTIC	SPEED_NAUTIC	LENGTH_NAUTIC

SHIPNO: ship identifier unique throughout the DB. It is part of the key in many other DB relations and part of the file name of files containing data of a given ship, for instance the intermediate files with file type BLOCK4, BLOCK5, BLOCK16.

SHNAME: ship's name. As it may happen that two different ships have the same name (more than often it so happens that a warship and a merchant ship bear the same name), this attribute cannot be a unique key.

FLAG: an abbreviation of the name of the flag country or an abbreviation of the owner's name. The values of this attribute are chosen from relation FLAGS.

SHTYPE: an abbreviation used to classify ships according to their type. The full caption is contained in relation TYPES.

LS: ship length, usually the length between perpendiculars, in m.

BS: ship breadth, usually molded breadth, in m.

TS: ship draft, usually molded draft at midship, for the design waterline, in m.

DS: ship depth, usually molded depth, in m.

HULLNO: has a significant value if the hull form or resistance data exist in the DB. Else HULLNO = "0" (see also the specification of relation HULL).

DISPLN: nominal displacement, correctly defined for the waterline described by TS. It may not be exactly the displacement resulting from hydrostatic calculations (analogy: a 1600 cu cm car has almost never a capacity of exactly 1600 cu cm). Therefore, this attribute is not redundant in the DB. DISPLN is measured in MN.

SPEEDM: maximum speed in knots. When two speeds may be defined, as for a naval vessel, the maximum (intermittent) speed should be entered here because this one determines the maximum installed power.

SPEEDC: cruising speed in knots. When two speeds may be defined, as for a naval vessel, the speed at which RANGE is calculated should be entered here because this one determines the endurance.

RANGE: the range in nautical miles for which the fuel quantity and the provisions are calculated.

Constraints:

- If LS < 3*BS warning

- If LS < BS error

- If TS > LS error

- If DS ≤ TS error

- If DISPLN > 1.03*LS*BS*TS warning

RELATION **HULL**

Application: defines the main characteristics of a hull for which the hull form and/or resistance/powering data are known. Relation HULL is one of the relations that generate the intermediate file BLOCK4.

HULLNO	SERIES	MARK	L	B	T

I05	A16	A20	F10.03	F10.03	F10.03
IDENTIFIER	IDENTIFIER	IDENTIFIER	LENGTH	LENGTH	LENGTH

HULLNO: hull identifier. More than one ship can be built with the same hull forms. Variants within the same series will have different SHIPNO identifiers but the same HULLNO. It is recommended that HULLNO be equal to SHIPNO of the first ship in a series of ships.

SERIES: defines the ship series or family to which the particular hull forms belong, or the institution where the forms were developed and/or tested. Examples: SERIES 60, SERIES 64, HSVA DESTROYER.

MARK: is the identifier of the particular hull within the given series. Example: C5 within HSVA DESTROYER.

L: hull length, usually the length between perpendiculars, possibly the waterline length. If the hull described by an individual tuple is the basin model of an actual ship, then looking for the same HULLNO in relation SHIPS we shall find a SHIPNO for which LS/L = BS/B = TS/T = λ = the geometrical similitude ratio.

For a hull derived by affine transformation we may have part or all of the following inequalities

$$LS \neq L\ ,\ BS \neq B\ ,\ TS \neq T\ ,\ LS/L \neq BS/B \neq TS/T\ .$$

B: hull beam, usually waterline beam at midship. Same comments as for L.

T: hull draft. Same comments as for L.

NOTES:

This relation may contain the data of a hull defined with the aid of scale factors that are not equal along all the three $x-$,$y-$ and $z-$ directions. This situation is allowed by the program ARCHIMEDES. Then L,B and T cannot be compared directly one with another. This is the reason why no constraints are imposed on these attributes in the same manner that they are imposed in relation SHIPS.

The FORTRAN formats of the main dimensions are not the same as in relation SHIPS. The reason is that the values stored in relation SHIPS are actual dimensions while those stored in relation HULL may be scaled dimensions. In our experience we met scale factors that required longer formats in order to achieve reasonable numerical precision.

RELATION **FLAGS**

Application: caption of the flag abbreviations used in relation SHIPS.

FLAG	FLGDES
A04	A36
IDENTIFIER	TEXT

FLAG: an abbreviation of the name of the flag country, or an abbreviation of the owner's name. For example:

ISR Israel
USA U.S.A.

"Flag" is an attribute in relation SHIPS.

FLGDES: a description of the flag country or of the owner.

RELATION **TYPES**

Application: caption of the ship type abbreviations used in relation SHIPS.

SHTYPE	TYPDES
A04	A56
IDENTIFIER	TEXT

SHTYPE: an abbreviation used to classify ships according to their type. For naval vessels and service craft the abbreviations follow mainly the list promulgated by the US Secretary of the Navy on 5 November 1979. For merchant ships most abbreviations are adopted from Lloyd's Ship Manager, World Order Book, No. 2, 1985.

As in Jane's Fighting Ships, the letter *"X"* may be added to existing abbreviations (not longer than 3 letters) to indicate a new class the characteristics of which have not been previously defined.

The abbreviation *NNNN* is adopted for unknown naval ship types. The abbreviation *MMM* is used for unknown merchant ship types. Examples:

DE Merchant deck cargo ship
PHM Guided missile patrol combatant (hydrofoil)

TYPDES: a text detailing the abbreviation.

HULL DATA

RELATION **LSPACES** (longitudinal spaces)

Application: describes some general properties of the whole ship, or of the so called "longitudinal spaces" processed in the program ARCHIMEDES. The latter are ship spaces the buoyancy of which can be added to, or deducted from the total buoyancy according to the configuration under study. See Fig. 6 for details.

HULLNO	SPACEN	KV	KF
------	------		
I05	I06	F05.02	F05.02
IDENTIFIER	IDENTIFIER	NONDIM	NONDIM

XMIDSH	CAMBER	APPEND	LSDES
F10.04	F10.02	F06.04	A30
LENGTH	NONDIM	NONDIM	TEXT

HULLNO: hull identifier as defined in relation HULL.

SPACENO:

SPACEN = 0 if the tuple refers to a whole ship,

$1 \leq$ SPACEN ≤ 99998 for an unsymmetrical longitudinal space

$-99998 \leq$ SPACEN ≤ -1 for a symmetrical longitudinal space.

This attribute represents the variable KZ in the program ARCHIMEDES. For more details see the guide to the program ARCHIMEDES [1], chapter 2.4.4.

KV: volume permeability; for a whole ship KV = -1

KF: surface permeability; for a whole ship KF = -1

XMIDSH: length coordinate of the midship section, in the coordinate system used for giving station longitudinal locations in relation STATNX (see Fig. 7).

CAMBER: camber ratio = deck breadth/camber. For flat decks assume CAMBER =1 000 000 .

APPEND: *appendage factor* defined by the following equalities

$$\Delta = (\textit{Displacement factor}) \cdot \nabla$$

$$Displacement\ factor\ =\ \gamma \cdot (\ APPEND\ )$$

where ∇ is the *molded volume of displacement*, and γ the *specific gravity* of the surrounding water. For instance, if $\gamma = 1.025$ and *APPEND* = 1.005 then

$$displacement\ factor\ =\ 1.025 \times 1.005\ =\ 1.030$$

and

$$\Delta\ =\ 1.030 \cdot \nabla\ .$$

By using an appendage factor, the buoyancy of shell and appendages is added to that of the molded hull. This factor shall be used whenever the above correction is not calculated separately.

Default values:

KV, KF = -1.0

CAMBER = 1 000 000

XMIDSH = 0.0

APPEND = 1.005.

Constraints:

- HULLNO must exist in relation HULL.

- If SPACEN = 0, then KV = -1, KF = -1

- ABS(KV) ≤ 1.0, ABS(KF) ≤ 1.0

- -L/2 ≤ XMID ≤ L/2

- 0 < CAMBER ≤ 1 000 000

NOTES:

1. It is recommended to take XMIDSH as 0.

2. The user working with this relation is strongly advised to read the guide to the program ARCHIMEDES [1]. The signs of KV and KF determine the sign of some calculated results. This is important in flooding calculations. For example, flooding calculations are performed in the ARCHIMEDES program by the lost buoyancy method. Therefore, for this program, when a space is lost in damage condition its KV and KF values must be positive.

RELATION **COMPART**

Application: stores the identifying numbers of the stations used to define a particular longitudinal space. In this relation the key is the whole tuple. COMPART is one of the relations that generate the intermediate file BLOCK4.

HULLNO	SPACEN	STATNO
------	------	------
I05	I06	I05
IDENTIFIER	IDENTIFIER	IDENTIFIER

HULLNO: hull identifier defined in relation HULL.

SPACENO:

SPACEN = 0 if the tuple refers to a whole ship. Else see relation LSPACES and the guide to the program Archimedes [1], chapter 2.4.4.

STATNO: identifying number of the offset station. For the same HULLNO no two stations shall have the same STATNO. Thus, for multiple pass hull descriptions (see guide to the program Archimedes [1], page 14) we shall assign different STATNO for the same station in different "passes". Each pass of the hull description shall be carried on in the positive x- direction, i.e. beginning aft and going on forwards.

Constraints:

- SPACEN must exist in relation LSPACES for the given HULLNO.

RELATION **STATNX** (station x-coordinate)

Application: contains the longitudinal position of transversal sections, in particular the offset stations used to define the hull surface in relation OFFSETS. See Fig. 7 for details. This is one of the relations that generate the intermediate file BLOCK4.

HULLNO	STATNO	STATX	STYP
------	------		
I05	I05	F10.04	A02
IDENTIFIER	IDENTIFIER	LENGTH	IDENTIFIER

STYPNO	ARCHI	WHIPI	SKI
F07.03	I01	I01	I01
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

HULLNO: hull identifier defined in relation HULL.

STATNO: the number of the offset station, with the constraint imposed in relation COMPART.

STATX: x- (longitudinal) coordinate of the station STATNO, measured in the system of coordinates with the origin defined by XMIDSH in relation LSPACES.

STYP: station type. Can assume one of the following values:

AE if it defines the aft extremity, that is the aft end of LOA;

AP if this is the geometrical station in the aft-perpendicular plane, that is a particular type of

ST station (see ST below);

BH for a bulkhead, that is a particular type of FR station (see FR below);

FE for the forward extremity, that is the forward end of LOA;

FP for the geometrical station (see below) situated in the forward perpendicular plane, that is a

particular type of ST station (see ST below);

FR for a structural frame;

KL for a transversal section used to define some keel particular;

ST for a geometrical station, that is a station appearing in the lines drawing.

STYPNO: identifying number within the particular type of stations. For instance, the frame number in the steel plan, if STYP = FR, the station number in the lines drawing, if STYP = ST.

ARCHI:

- if ARCHI = 1 then this station will be used in hydrostatic calculations (for instance, it would be included in BLOCK 4 for calculations by the ARCHIMEDES program);

- ARCHI = 0 otherwise.

WHIPI:

WHIPI = 1 if the station is used in whipping calculations;

WHIPI = 0 otherwise

SKI:

SKI = 1 if the station is used in seakeeping calculations

SKI = 0 otherwise

Constraints:

For given HULLNO, STATNO must exist in relation COMPART.

Stations for which ARCHIN =1 shall fulfill a constraint imposed by Simpson's rule, that is there must be pairs of equal station spacings.

RELATION **OFFSETS**

Application: stores the coordinates of points that describe sectional contours, and completes the description of the hull surface. See Fig. 8 for details.

HULLNO	STATNO	POINTN	POINTZ
------	------	------	
I05	I05	I03	F10.04
IDENTIFIER	IDENTIFIER	IDENTIFIER	LENGTH

POINTY	PTYPE	PTYPNO	ARCHIN	SKIN
F10.04	A01	I02	I01	I01
LENGTH	IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

HULLNO: hull identifier defined in relation HULL.

STATNO: the number of the offset station with the constraint imposed in relation COMPART.

POINTN: identifier of the offset point described by the tuple. POINTN is part of the key and is assigned sequentially within the same station. This is not the point number used in BLOCK 4 of the program ARCHIMEDES (see users guide [1]).The interface program PBLOCK4 derives the ARCHIMEDES point number from this attribute.

Points should be described beginning at the keel and continuing upwards, to the deck at side, possibly the deck-at-center line.

It is recomended to initially assign POINTN in steps of 5, e. g. 0, 5, 10, 15, ... Thus, if necessary, it would be possible to insert further points in the proper order.

POINTZ: Z-coordinate of the point POINTN.

POINTY: Y-coordinate of the point POINTN.

PTYPE: describes the type of longitudinal line to which the point belongs. The possible values of this attribute are;

B - buttock;

C - deck at center;

D - diagonal;

K - knuckle (chine);

S - deck at side;

W - waterline.

PTYPNO: an identifying number assigned sequentially within the set of lines having the same PTYPE value, for a given HULLNO (see Fig. 8).

ARCHIN:

- if ARCHIN = 1 then this point will be included in BLOCK 4 for calculations by the ARCHIMEDES program;

- ARCHIN = 0 otherwise.

SKIN:

SKIN = 1 if the point is included in seakkeping calculations (for Frank close fit);

SKIN = 0 otherwise

Default values:

- PTYPE = 'W'

- PTYPNO = 1

- ARCHIN, SKIN = 0

Constraints:

- HULLNO must exist in relation HULL.

- At given HULLNO, STATNO must exist in relations COMPART and STATNX.

- If POINTN = 0, then POINTY = 0.0000 .

NOTES:

1. Buttock lines (PTYPE = 'B') shall be assigned increasing numbers from CL towards the shell.

2. Do not give the CL point if the camber shape at this station is parabolic.

3. Waterlines (PTYPE = 'W') shall be assigned increasing numbers from the baseline upwards.

RELATION **FREEBRD** (freeboard)

Defines points of a line, possibly immaginary, the position of which above the waterline must be checked in case of flooding. For instance, for merchant ships, or for ships designed according to US Navy specifications, the points defined in this relation describe the margin line situated 3 inches below the bulkhead deck. For a naval vessel designed to Federal Germany specifications, the points in this relation generally describe the deck at side line. In certain cases it may be interesting to describe lines that shall not submerge because they include non-watertight openings such as vents or overflow pipes.

SHIPNO	FBNO	FBX	FBY	FBZ	FBDES
---	---				
I05	I03	F07.03	F07.03	F07.03	A40
IDENTIFIER	IDENTIFIER	LENGTH	LENGTH	LENGTH	TEXT

SHIPNO: ship identifier throughout the database, defined in relation SHIPS.

FBNO: point identifier.

FBX: x-coordinate measured from AP, positive forward.

FBY: x-coordinate measured from CL, positive to starboard.

FBZ: z-coordinate measured from BL, positive upwards.

FBDES: explanations such as a description of the respective point.

CONTOUR DATA

RELATION **CONDES** (contour description)

Application: this relation assigns to each SHIPNO a longitudinal contour described in relation CONTOUR and a transverse contour described in relation TCONT.

SHIPNO	CNO	CONTYP	CONNAM
---	---	---	
I05	I02	A01	A36
IDENTIFIER	IDENTIFIER	IDENTIFIER	TEXT

SHIPNO: ship identifier throughout the database, defined in relation SHIPS.

CNO: contour number, key attribute in relations CONDES, where it is equal to CONTNO, and TCONT.

CONTYP:

CONTYP = L for a longitudinal contour described in relation CONTOUR

CONTYP = T for a transverse contour defined in relation TCONT

CONNAM: a description of the respective contour, for instance "3 container tiers on deck".

RELATION **CONTOUR**

Application: digital description of the ship contour above deck, that is of the ship projection on a plane parallel to the centerline plane. It does not include the contour under the deck as this can be obtained by selecting in relation OFFSETS the points for which POINTY = 0. These points and relation CONTOUR generate the intermediate file SCONTOUR used for wind arm calculations. Fig. 9 exemplifies an application of this relation.

SHIPNO	CONTNO	CPOINN	CPOINX	CPOINZ
------	------	------		
I05	I02	I05	F10.03	F10.03
IDENTIFIERI	IDENTIFIER	IDENTIFIER	LENGTH	LENGTH

SHIPNO: ship identifier defined in relation SHIPS.

CONTNO: number of the contour to be used in sail area calculations. It is defined in relation LDCASES as a function of the loading case number LCASE. Several longitudinal contours can be defined for one SHIPNO.

CPOINN: identifier of the contour point, used in order to make the key unique.

CPOINX: x-coordinate of point CPOINN, measured from AP and positive forwards.

CPOINZ: z-coordinate of point CPOINN, measured from BL and positive upwards.

NOTE:

Why cannot CPOINX be part of a unique key? Because we can easily imagine contour segments in which two points have the same x-coordinate, but different z-coordinates. For similar reasons CPOINZ cannot be part of a unique key. In fact CPOINN is a *surrogate* key attribute which can be assigned by the system.

RELATION **TCONT** (transverse contour)

Application: defines a transversal contour which may be used in calculations of wind resistance. Fig. 10 exemplifies an application of this relation.

SHIPNO	TCONTN	XCONT	TPOINN	TPOINY	TPOINZ
---	---	---	---		
I05	I02	F10.03	I05	F10.03	F10.03
IDENTIFIER	IDENTIFIER	LENGTH	IDENTIFIER	LENGTH	LENGTH

SHIPNO: ship identifier defined in relation SHIPS.

TCONTN: identifier of transverse contour. Several transversal contours may be defined for one SHIPNO. Relation LDCASES assigns to each load case the appropriate transverse contour. Relation CONDES contains the description of each transverse contour.

XCONT: x – coordinate, measured from AP forwards, of the transverse plane that contains the contour.

TPOINN: identifier of the contour point. This attribute assures that each tuple will be identified by a unique key.

TPOINY: y – coordinate of the contour point, measured from the CL plane, positive to starboard.

TPOINZ: z – coordinate of the contour point, measured from the baseline, positive upwards.

SUBDIVISION DATA

RELATION **COMPNAM**

Application: defines names of ship compartments. The information contained in this relation serves for a better description of the ship and for assigning each compartment one of the permeabilities listed in relation FLDPERM. These permeabilities are further used in damage calculations. Fig. 11 exemplifies an application of this relation.

SHIPNO	COMPNO	COMPNM	COMTYP
---	---		
I05	I03	A20	A02
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

SHIPNO: ship identifier defined in relation SHIPS.

COMPNO: compartment number defined in relations SUBSPCE and LSTSPCE.

COMPNM: compartment name.

COMTYP: serves in choosing the appropriate permeability. Examples of COMTYP values:

AM ammunition store

LS	lost space
RG	regular compartment
ST	store room
TK	tank

Lost space means a space the buoyancy of which shall be deducted in calculations. By defining such spaces it may be easier to describe hull forms affected by irregularities such as a moonpool or a non-watertight compartment. For example, the hull of a service vessel provided with a moonpool may be described by regular offsets, ignoring the moonpool. The moonpool shall be then described separately and defined as being of type LS. The system will assign such a permeability that the buoyancy of the moonpool will be deducted in calculations.

RELATION **SUBSPCE**

Application: this relation describes the subdivision of the ship. Figures 12 and 13 exemplify an application of relation SUBSPCE.

SHIPNO	COMPNO	SSPCNO	SPCTYP
------	------	------	
I05	I03	I02	I01
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

XA	XF	ZAL	ZFL
F07.03	F07.03	F06.03	F06.03
LENGTH	LENGTH	LENGTH	LENGTH

ZAU	ZFU	YAL	YFL	YAU
F06.03	F06.03	F06.03	F06.03	F06.03
LENGTH	LENGTH	LENGTH	LENGTH	LENGTH

SHIPNO: ship identifier defined in relation SHIPS.

COMPNO: compartment, tank or hold identifier. For computational reasons it may be necessary to divide one compartment into several standard-type subspaces, for example as described in the guide to the program ARCHIMEDES [1], Chapter 2.4.5. See also Fig. 12.

SSPCNO: subspace identifier. The concatenation of COMPNO and SSPCNO yields an identifier of a subspace belonging to the compartment COMPNO.

SPCTYP: standard space type, for instance acc. to Fig. 5 in the guide to the program ARCHIMEDES [1].

XA: x-coordinate of the aft transversal bulkhead (or imaginary transverse plane) delimiting the subspace.

XF: x-coordinate of the forward transversal bulkhead (or imaginary transverse plane) delimiting the subspace.

ZAL: z-coordinate of the delimiting lower deck (or imaginary horizontal plane), at the aft transversal bulkhead.

ZFL: z-coordinate of the delimiting lower deck (or imaginary horizontal plane), at the forward transversal bulkhead.

ZAU: z-coordinate of the delimiting upper deck (or imaginary horizontal plane), at the aft transversal bulkhead.

ZFU: z-coordinate of the delimiting upper deck (or imaginary horizontal plane), at the forward transversal bulkhead.

YAL: y-coordinate of the delimiting longitudinal bulkhead (or imaginary longitudinal plane) at the lower point aft.

YFL: y-coordinate of the delimiting longitudinal bulkhead (or imaginary longitudinal plane) at the lower point forward.

YAU: y-coordinate of the delimiting longitudinal bulkhead (or imaginary longitudinal plane) at the upper point aft.

The "longitudinal" plane defined by the three cordinates YAL, YFL, YAU is not necessarily parallel to the centerline plane, but may be inclined.

Default values:

- VPERM, SFPERM = 1

Constraints:

- $-1.0 \leq$ VPERM, SPERM ≤ 1.0

The user is advised to adopt some convention when assigning values to COMPNO and SSPCNO. For example: the first two digits of COMPNO mark the compartment number.

NOTES:

1. Usually VPERM = SFPERM.

2. See relation COMPNAM for naming.

FLOODING DATA

RELATION **FLDCASE** (flooding case)

Application: lists the flooding cases of interest for a given ship. Thus, if for a given SHIPNO, *n* flooding cases must be considered, this relation will contain *n* records *(tuples)* beginning with the given SHIPNO value.

SHIPNO	FDCASE	FDDES
------	------	
I05	I03	A36
IDENTIFIER	IDENTIFIER	TEXT

SHIPNO: ship identifier defined in relation SHIPS.

FDCASE: identifier of flooding case.

FDDES: a description of the respective flooding case, for example "stern flooding".

RELATION **FLDCOMP** (flooded compartments)

Application: for a given SHIPNO lists the compartments flooded in each flooding case. Thus, if for a given SHIPNO, *n* flooding cases must be considered, and for a given flooding case *i* there are m_i flooded compartments, this relation contains $\sum_i m_i$ lines *(tuples)* for the given SHIPNO value.

SHIPNO	FDCASE	COMPNO
------	------	------
I05	I03	I03
IDENTIFIER	IDENTIFIER	IDENTIFIER

SHIPNO: ship identifier defined in relation SHIPS.

FDCASE: identifier of flooding case, defined in relation FLDCASE.

COMPNO: identifier of flooded compartment.

RELATION **FLDPERM** (flooding permeabilities)

Application: assigns a permeability value for each ordered pair LDTYPE, COMTYP, where LDTYPE is taken from relation LDCASES and COMTYP from relation COMPNAM. Thus, floding calculations may

be performed with automatically-choosen, correct permeability values.

COMTYP	LDTYP	VPERM
A02	A02	F05.03

COMTYP: compartment type defined in relation COMPNAM.

LDTYPE: assigned in relation LDCASES for each pair SHIPNO, LCASE. For example:

COMTYP	LDTYPE	VPERM
AM	FL	0.800
AM	HL	0.870
ST	FL	0.600
ST	HL	0.700

MASS DATA

RELATION **WEIGSYS** (weight group classification system)

Application: defines the weight classification system adopted for a particular ship.

SHIPNO	SYSTEM
I05	A20
IDENTIFIER	IDENTIFIER

SHIPNO: ship identifier, defined in relation SHIPS

SYSTEM: the name of the classification system used for the items of ship SHIPNO. Examples: SWBS, MARAD, SFI, PRELIMINARY.

RELATION **CLASSYS** (classification system)

Application: describes the weight groups of each classification system encountered in the DB.

SYSTEM	ITEM	ITYPE	DETAIL
A20	A07	A01	A40
IDENTIFIER	CLASS	IDENTIFIER	TEXT

SYSTEM: the name of the classification system, e.g. MARAD, SWBS, SFI, PRELIMINARY.

ITEM: the code of the particular weight group. If the system involved uses less than 7 digits, the significant digits should be left-adjusted and the rest filled in with zeroes. The user may change the zeroes into significant digits, according to his own needs, in order to distinguish between different items belonging to the same weight group. If SYSTEM = PRELIMINARY, the user may complete his own details according to his needs. For the SWBS system the 7 digits follow the schema AAABBCC, where AAA is the main classification, BB gives the specific detail, and CC is the serial number.

ITYPE: = R for regular weight items, = P for naval payload-weight items, = N for non-weight items. This attribute allows weight calculating programs to automatically pick up weight items and to check up mass-item lists. This attribute is especially relevant for naval vessels. Fot merchant ships it can take only the R and N values.

DETAIL: a description of the weight group.

Examples of records (tuples) in relation CLASSYS:

SWBS 113000 R INNER BOTTOM

SWBS 201000 N GENERAL ARGMNT., PROPULSION DRAWINGS

SWBS 460000 P SONAR

SWBS F63000 R CARGO FUELS, LUBRICANT

RELATION **LIGHTWT**

Application: this is actually a "weight book" in which the mass properties of lightweight items are stored. For example: shell plates, stringers, brackets, pieces of furniture, engines, valves. This relation is used to generate the intermediate files BLOCK16, IN5D, INNV5D, INSCORES, SUMMARY, WGHTOTAL.

SHIPNO	ITEM	QTY	DSMASS	XCG	YCG
I05	A07	I04	F09.03	F10.03	F10.03
IDENTIFIER	CLASS	INTEGER	MASS	LENGTH	LENGTH

ZCG	XAFT	XFWD	LDNAME	LDORIG
F10.03	F10.03	F10.03	A30	A01
LENGTH	LENGTH	LENGTH	TEXT	IDENTIFIER

SHIPNO: ship identifier, as defined in relation SHIPS.

ITEM: code number of the particular load, according to the classification system defined in relation WEIGSYS. See relations CLASSYS and BULKLD for more explanations.

QTY: the number of pieces that share the same attribute values. For example, if two identical valves are located in such a manner that the attributes XCG, YCG, XAFT, XFWD can be also considered identical conserves the total mass and center of gravity but falsifies the moment of inertia.

DSMASS: mass of one piece.

XCG: longitudinal coordinate of the center of gravity, measured from AP, positive forwards. If QTY > 1, this is the coordinate of the common center of gravity of all units described by the same tuple.

YCG: transverse coordinate of the center of gravity, measured from CL, positive to starboard. If QTY > 1 , see remark for XCG.

ZCG: height coordinate of the center of gravity, measured from BL, positive upwards. If QTY > 1 , see remark for XCG.

XAFT: X-coordinate of the aftmost point of the load, measured in the same manner as XCG.

XFWD: X-coordinate of the foremost point of the load, measured in the same manner as XCG.

LDNAME: load details given by the Naval Architect for his convenience.

LDORIG: is equal to C for Calculated, E for Estimated, A for Actual. This attribute qualifies the degree of data uncertainty.

Default

-LDORIG= E

- QTY = 1

- YCG = 0.000

- XAFT = XFWD = XCG if XCG is known and XAFT, XFWD unknown

(these default values model the weights as concentrated, while in reality they are distributed longitudinally. Thus, it may be assumed that the calculated bending moment will be larger and the results will be on the safe side.)

- XCG = YCG = ZCG = XAFT = XFWD = 0 if no data about the centers of gravity are available.

Constraints:

- SHIPNO must exist in relation SHIPS.

- ITEM must exist in relation CLASSYS, for the system defined in relation WEIGSYS at the given SHIPNO. It is possible to have only the first digits taken from a classification system, and to use the rightmost digits according to a user defined convention.

- QTY*MASS < DISPLN/g , where DISPLN is taken from relation SHIPS for the same SHIPNO and g is the acceleration of gravity.

- $-0.25*LS < XAFT \leq XCG < XFWD \leq 1.25*LS$

- If YCG > BS/2 then warning, where BS is read from relation SHIPS, for the given SHIPNO

RELATION **LDCASES**

Applications: assigns to each load-case number an explanation and contour numbers.

SHIPNO	LCASE	LDTYPE	CONTNO	TCONTN	LEGEND
------	------				
I05	I02	A02	I02	I02	A36
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER	TEXT

SHIPNO: ship identifier defined in relation SHIPS.

LCASE: load case number, according to some convention accepted by the Naval Architect. This attribute shall be defined exactly as in relations BULKLD and DISCLD where it is part of the key.

LDTYPE: serves to choose the appropriate permeability in relation FLDPERM. Example values:

FL full load
HL half load

LEGEND: a characterization of the load case, for example lightship, ballast arrival, ballast departure, full load arrival, mission ready.

CONTNO: the number of the contour to be used in sail area calculations. It is defined in this relation and is part of the key in relation CONTOUR.

TCONTN: the number of the transverse contour to be used in transverse area calculations. It is defined in this relation, and is part of the key in relation TCONT.

Default values:

- CONTNO = 1

- TCONTN, LEGEND= blank

RELATION **DISCLD** (discrete load)

Application: this relation defines the loading of the vessel with discrete deadweight/payload loads such as containers, vehicles, airplanes or helicopters, for each load-case in part.

SHIPNO	LCASE	ITEM	QTY	DSMASS	XCG
------	------	------			
I05	I02	A07	I04	F09.03	F10.03
IDENTIFIER	IDENTIFIER	CLASS	INTEGER	MASS	LENGTH

YCG	ZCG	XAFT	XFWD	LDNAME	LDORIG
F10.03	F10.03	F10.03	F10.03	A30	A01
LENGTH	LENGTH	LENGTH	LENGTH	TEXT	IDENTIFIER

SHIPNO: ship identifier defined in relation SHIPS.

LCASE: load-case number defined according to some convention, as mentioned for relation BULKLD.

ITEM: code number of the particular load, according to the classification system defined in relation WEIGSYS. See relation BULKLD for more explanations.

QTY: the number of pieces that share the same attribute values. For example, if two identical containers are located in such a manner that the other attributes (XCG, YCG, XAFT, XFWD) can be also considered identical, then QTY = 2. Attention is drawn to the fact that describing several items together, under QTY $\neq$ 1, conserves the total mass and center of gravity, but yields false values of the moments of inertia.

DSMASS: load mass.

XCG: longitudinal coordinate of the center of gravity, measured from AP, positive forwards.

YCG: transverse coordinate of the center of gravity, measured from CL, positive to starboard.

ZCG: height coordinate of the center of gravity, measured from BL , positive upwards.

XAFT: x-coordinate of the aftmost point of the load, measured in the same system as XCG.

XFWD: x-coordinate of the foremost point of the load, measured in the same system as XCG.

LDNAME: details of the discrete-load item, given by the Naval Architect for his convenience.

LDORIG: equals C for Calculated, E for Estimated, A for Actual. This attribute qualifies the degree of data uncertainty.

- YCG = 0

- WTYPE = E

- LDNAME = blank

Constraints:

- SHIPNO must exist in relation SHIPS.

- ITEM, see constraint in relation BULKLD.

- MASS < DISPLN/g, where DISPLN is taken from relation SHIPS for the same SHIPNO, and g is the acceleration of gravity.

- -1.25*LS < XAFT ≤ XCG ≤ XFWD < 1.25*LS

- if YCG > BS/2 warning, where BS is read from relation SHIPS for the given SHIPNO.

RELATION **BULKLD**

Application: This relation defines the loading of the vessel with deadweight/payload bulk loads (liquids, grains, ore) for each load case in part. The relation tells how much of a given tank/hold is filled, and with what kind of material it is filled. BULKLD does not directly indicate load masses. These can be calculated as DENSITY*FILLING*CAPACITY and therefore would be redundant in a relational data base.

SHIPNO	LCASE	ITEM	COMPNO	FILLIN	DENSIT
I05	I02	A07	I03	F04.02	F06.03
IDENTIFIER	IDENTIFIER	CLASS	IDENTIFIER	NONDIM	DENSITY

SHIPNO: ship identifier defined in relation SHIPS.

LCASE: load-case number defined according to some convention accepted by the Naval Architect.

ITEM: the initial substring of ITEM, further called *significant part,* is the code number of a particular load, according to some classification system such as SWBS, MARAD or the SFI Group System. The particular classification system employed for the ship SHIPNO is declared in relation WEIGSYS. Knowing attribute SYSTEM from relation WEIGSYS and attribute ITEM from relation BULKLD, one can find its meaning in relation CLASSYS.

The remaining substring of ITEM is an augmentation of the classification system plus/or an *identification code* defined by the DB user.

COMPNO: compartment (tank,hold) identifier defined in relation COMPNAM.

FILLIN: filled capacity/total capacity ratio for the respective tank or hold.

DENSIT: the density of the bulk load, in units of t/m^3.

Default values:

- DENSIT = 1.0.

Constraints:

- SHIPNO must exist in relation SHIPS, and for the corresponding HULLNO there must exist tuples in relations HULL, LSPACES, COMPART, OFFSETS, STATNX, and SUBSPCE.

- COMPNO must exist in relation SUBSPCE, for the given SHIPNO.

- $0.0 \leq$ FILLIN ≤ 1.0

- $0.0 <$ DENSIT ≤ 22.5

- Warning: The significant part of ITEM should exist in relation CLASSYS, for the SYSTEM defined in relation WEIGSYS for the given SHIPNO. The significant part length is dependent on the classification system chosen.

Note

It usually happens that one and the same load is common to several load cases. A tuple for that load should then appear for each load case involved. One key attribute, LCASE, will differ from tuple to tuple and their keys will thus uniquely define them.

RELATION **DEADWGT**

Application: stores details of deadweight/payload weights for ships for which SYSTEM = PRELIMINARY in relation WEIGHSYS. For such ships, usually in an incipient stage of design, it would not be practical to define tuples in relation BULKLD. Therefore, fuel, water, grain cargo etc. could be entered here with very little detail. This relation may be used to generate intermediate files SUMMARY and WGHTOTAL (see [22] and [23]).

SHIPNO	LCASE	ITEM
---	---	---
I05	I02	A07
IDENTIFIER	IDENTIFIER	CLASS

DWMASS	XCG	ZCG	LDORIG
F09.03	F10.03	F10.03	A01
MASS	LENGTH	LENGTH	IDENTIFIER

SHIPNO: ship identifier defined in relation SHIPS.

LCASE: load case number. See comments in relation BULKLD.

ITEM: the code of the weight group, according to a rudimentary classification system stored in relation CLASSYS.

DWMASS: the mass of the weight group.

XCG: longitudinal (i.e. x-) coordinate of the center of gravity measured from AP, positive forwards.

ZCG: height (i.e. y-) coordinate of the center of gravity, measured from BL, positive upwards.

LDORIG: equals C for Calculated, E for Estimated and A for Actual. This attribute qualifies data uncertainty.

Constraints:

- $-0.25*LS \leq XCG \leq 1.25*LS$ where LS is taken from relation SHIPS,

- For any SHIPNO appearing here, SYSTEM = PRELIMINARY in relation WEIGSYS.

NOTE:

YCG does not appear here because it is assumed in the preliminary design stage that the ship is balanced transversally.

RESISTANCE AND PROPULSION DATA

RELATION **MODEL**

Application:

- describes the model used to obtain the data in relation RESIST;

- assigns this model to either a ship in relation SHIPS, if such a relationship exists, or to a hull form defined in relation HULL.

MODNO	MODVAR	MODREF	BASIN
I05	A02	A01	A20
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

BASNO	FRLENG	FRTYPE	MODDES
A10	F07.03	A01	A30
IDENTIFIER	LENGTH	IDENTIFIER	TEXT

MODNO: model identifier within the DB. It is the key attribute of the relation. If the model can be related to a ship described in the data base, it is recommended to assign a MODNO value equal to SHIPNO of the first ship in the respective series, otherwise MODNO = HULLNO.

MODVAR: identifies model variants of the same MODNO. For instance, model variants can be models at different scales of the same ship, one for resistance tests, another one for seakeeping tests, etc.

MODREF:

MODREF: = "S" if MODNO is a ship number;

MODREF = "H" if MODNO is a hull number.

BASIN: the name of the test basin that carried on the model tests. Alternatively, it may be the name of the shipyard that carried out the sea trials, or the name of the measured mile location.

BASNO: the model number assigned by the test basin and contained in the test report.

FRLENG: the characteristic length with which FMODEL in relation RESIST is calculated. It usually is LWC for displacement type vessels, B for planning hulls. It may also be $\nabla^{1/3}$.

FRTYPE: defines the nature of FRLENG. Specifically:

FRTYPE = "L" when FRLENG = LWC (waterline length);

FRTYPE = "B" when FRLENG = B ;

FRTYPE = "V" when FRLENG = $\nabla^{1/3}$.

MODDES (model description): comments, such as "Bare hull", "Self propulsion" a. s. o.

Default values:

MODVAR = 0 ;

BASIN, BASNO, MODDES = blank ;

Constraints:

the value of MODNO must exist in either relation HULL or relation SHIPS ;

Notes.

A length, rather than a scale is used to define the model size. Test data stored in relation RESIST are nondimensionalized by this length.

RELATION **RUNDATA**

Application: details the conditions under which tests were run.

MODNO	MODVAR	SERNO	RUNDES	DRAFT
------	------	------		
I05	A02	I03	A20	F06.03
IDENTIFIER	IDENTIFIER	IDENTIFIER	TEXT	LENGTH

TRIM	WTKIND	CA	TEMP	DATE
F05.03	A01	F07.05	F04.01	A08
LENGTH	IDENTIFIER	NONDIM	TEMPERATURE	TEXT

MODNO: model identifier within the DB, defined in relation MODEL.

MODVAR: identifies model variants of the same MODNO, as explained in relation MODEL.

SERNO: number of test run, preferably equal to that in the test report.

RUNDES: a description of the test run. For instance: "New propeller model", "New sonar dome".

DRAFT: model draft in meters.

TRIM: static trim if the model can trim freely, or forced trim if the model is run at an imposed trim angle (in meters). Trim by the stern > 0 ; this means

$$TRIM = T_{AFT} - T_{FWD} \quad .$$

TEMP: water temperature during test, in centigrade degrees (Celsius).

WTKIND:

WTKIND = "F" if the test was carried on in fresh water;

WTKIND = "S" if the test was performed in salt water.

CA: correlation allowance as used by the model basin in its report.

DATE: test date written as YEAR.MONTH.DATE in the format *XX.XX.XX* .

Default values:

SERNO assigned sequentially for constant MODNO: 1, 2, 3, ...;

RUNDES, DATE blank;

TRIM = 0 ;

WTKIND = "F" ;

CA to be calculated by a built-in function of the application program.

Constraints:

MODNO and MODVAR must exist in relation MODEL, and all constraints described in that relation must be fulfilled.

$0 \leq \text{TEMP} \leq 30$.

WTKIND $\in$ {F, S} .

RELATION **RESIST**

Application: stores the nondimensional results of model tests. This relation is used to calculate ship resistance and powering.

MODNO	MODVAR	SERNO	FMODEL
---	---	---	---
I05	A02	I03	F07.04
IDENTIFIER	IDENTIFIER	IDENTIFIER	NONDIM

CRS	DTRIM	DSINK
F10.07	F11.08	F07.04
NONDIM	ANGLE	NONDIM

MODNO: model identifier within the DB, defined in relation MODEL.

SERNO: number of test series, defined in relation RUNDATA.

FMODEL: Froude number at which the test was run and defined as

$$FMODEL = \frac{V}{\sqrt{g \cdot FRLENG}}$$

where V is the test speed in m/s , g the acceleration of gravity and $FRLENG$ a model characteristic length stored in relation MODEL.

CRS: coefficient of residual resistance defined as

$$\frac{R_R}{0.5 \cdot WTRHO \cdot V^2 \cdot S}$$

where

R_R is the residuary resistance, i. e. $R_R = R_T - R_F$;

WTRHO is the water density stored in relation WATDENS ;

V is the model speed ;

S is the wetted surface .

Unless otherwise mentioned, CRS is the scaled up value for the appended ship.

DTRIM: dynamic trim in radians. It is the trim that a free trimming model will assume during the test run. The sign of DTRIM is defined in the same manner as the sign of attribute TRIM in relation RUNDATA, i. e. trim by the stern is positive.

DSINK: nondimensional dynamic sinkage, i. e. the vertical displacement of the center of gravity during the test run. DSINK is considered positive if the center of gravity rises. DSINK is nondimensionalized by dividing by FRLENG, an attribute defined in relation MODEL.

Default values:

DTRIM, DSINK = 0 .

Constraints:

MODNO must exist in relation MODEL; RUNNO must exist in relation RUNDATA;

0 < FMODEL < 5.0 ;

Note.

As the key attribute FMODEL and the test results *CRS* , *DTRIM* and *DSINK* are nondimensionalized by *FRLENG* , it is sufficient to input the desired value of this attribute to a program for calculating resistance, in order to generate results at ship scale.

RELATION **WETSURF** (wetted surface)

Application: stores wetted surface values of models or of ships, if offsets are not available in order to calculate directly the wetted surface. The information in this relation can be obtained either from test basin publications or by approximate formulae.

MODNO	MODVAR	SERNO	S
I05	A02	I03	F10.04
---	---	---	
IDENTIFIER	IDENTIFIER	IDENTIFIER	NONDIM

MODNO: model identifier in the DB, defined in relation MODEL.

MODVAR: identifies model variants of the same MODNO, as explained in relation MODEL.

SERNO: identifier of the test series, defined in relation RUNDATA.

S: wetted surface at the respective draft, nondimensionalized by dividing by $FRLENG^2$. For FRLENG see relation MODEL.

Constraints:

MODNO and MODVAR must exist in relation MODEL, SERNO must exist in relation RUNDATA.

Relation **KVISC** (kinematic viscosity)

Application: This relation is a table of fresh- and salt-water kinematic viscosities, for temperatures ranging between 0 and 30.9 degrees centigrade (Celsius), at 0.1 degree intervals. These values are useful in hydrodynamic calculations such as ship resistance prediction from model tests. The stored density values are reproduced from Rawson and Tupper, "Basic Ship Theory", 2nd edition, Longman, 1976. The salt-water densities correspond to a salinity of 3.5 per cent.

WTKIND	TEMP	KINVIS
------	----	
A01	F04.01	F07.05
IDENTIFIER	TEMPERATURE	KINEM_VISCOSITY

WTKIND: identifies the water kind involved:

F describes fresh water;

S describes salt water of 3.5 per cent salinity.

TEMP: water temperature in degrees centigrade (Celsius).

KINVIS: kinematic viscosity in units of $m^2s^{-1}\times10^6$

Relation **WATDENS** (water density)

Application: This relation is a table of fresh- and salt-water densities, for temperatures ranging between 0 and 30 degrees centigrade (Celsius). These values are useful in hydrodynamic calculations such as ship resistance prediction from model tests. The stored density values were reproduced from Rawson and Tupper, "Basic Ship Theory", 2nd edition, Longman, 1976. The salt-water densities correspond to a salinity of 3.5 per cent. According to the above mentioned authors, the last decimal figure is doubtful

WTKIND	TEMP	WTRHO
-------	-------	
A01	I02	F07.05
IDENTIFIER	TEMPERATURE	DENSITY

WTKIND: identifies the water kind involved:

F describes fresh water;

S describes salt water of 3.5 per cent salinity.

TEMP: water temperature in degrees centigrade (Celsius).

WTRHO: water density in units of t/m^3 (ton mass / cu meter).

RELATION **SELFPRO**

Application: stores the results of self-propulsion tests.

MODNO	MODVAR	SERNO	PROPNO	PROPVA	POD
------	------	------	------	------	------
I05	A02	I03	I06	I03	F05.03
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER	NONDIM

SPEED	RPM	TFAC	WFAC	ETHAR

F07.03	F06.01	F07.04	F07.04	F06.04
SPEED	ROT_SPEED	NONDIM	NONDIM	NONDIM

MODNO: model identifier within the DB, defined in relation MODEL.

MODVAR: identifies model variants of the same MODNO. It is defined in relation MODEL.

SERNO: test series number (see also relation RUNDATA).

PROPNO: propeller identifier in the DB, defined in relation PROPSER.

PROPVA: identifier of a constructive propeller, that is a propeller with a given diameter, within the series SERNO which defines nondimensional propellers. This attribute is defined in relation PROPPAR.

POD: actual pitch-to-diameter ratio, P/D , used during test. See comments in relation PROPDAT.

SPEED: model speed, m/s.

RPM: rotational speed of propeller shaft, rpm.

TFAC: thrust deduction factor, defined by

$$t = 1 - \frac{R_T}{T}$$

where R_T is total resistance and T propeller thrust.

WFAC: wake fraction, defined by

$$w = \frac{V - V_A}{V_A}$$

where V is ship speed and V_A speed of advance.

As known, hull efficiency is defined as

$$\eta_H = \frac{R_T V}{T V_A} = \frac{1 - t}{1 - w}$$

ETHAR: relative rotative efficiency, defined by

$$\eta_R = \frac{\eta_B}{\eta_o}$$

where η_B is the *efficiency behind the hull* and η_o is the *open water efficiency.*

Default values:

$$ETHAR = 1.0$$

RELATION **PROPINS** (propeller installation)

Application: stores details on propeller arrangement behind an actual ship.

SHIPNO	SHFTNO	PROPNO	PROPVA	ROT
-----	-----			
I05	I01	I06	I03	A02
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER	TEXT

SHIPNO: ship identifier within the DB, defined in relation SHIPS.

SHFTNO: shaft identifying number. Numbering goes from starboard to port, that is the extreme-starboard shaft is assigned SHFTNO = 1 and so on.

PROPNO: propeller identifier within the DB, as defined in relation PROPSER.

PROPVA: (propeller variant) identifies a constructive propeller within its series. Defined in relation PROPPAR.

ROT: defines the sense of rotation:

ROT = RH when right-handed;

ROT = LH when left-handed.

Constraints:

SHIPNO must exist in relation SHIPS, PROPNO must exist in relation PROPPAR

RELATION **ENGINES**

Application: a catalog of engines and their principal characteristics. The performances of diesel engines are stored in relations DIESSFC and DIESDAT; the performances of gas turbines, in relations TURBSFC and TURBDAT.

ENGNO	EMAKE	ETYPE	EKIND

I07	A15	A20	A03
IDENTIFIER	IDENTIFIER	IDENTIFIER	IDENTIFIER

MPOWR	MRPM	CPOWR	CRPM	EREM
F08.01	F06.01	F08.01	F06.01	A30
POWER	ROT_SPEED	POWER	ROT_SPEED	TEXT

DSMASS	LENGT	WIDTH	HEIGH
F09.03	F06.03	F06.03	F06.03
MASS	LENGTH	LENGTH	LENGTH

ENGNO: engine identifier in the DB. This attribute is the key of the relation.

EMAKE: manufacturer identifier.

ETYPE: engine identifier in manufacturer's catalog.

EKIND: engine type, for instance:

DIS = diesel engine

GT = gas turbine

ST = steam turbine.

MPOWR: maximum intermitent engine rating, kW.

MRPM: rotational speed, rpm, at MPOWR.

CPOWR: maximum continous rating, kW.

CRPM: rotational speed, rpm, at CPOWR.

EREM: notes, for example a description of the conditions under which the engine tests were run.

DSMASS: engine mass in t mass (SI unit).

LENGT: engine length in m.

WIDTH: engine width in m.

HEIGH: engine height in m.

Constraints:

$$MPOWR \geq CPOWR > 0$$

$$MRPM \geq CRPM > 0$$

$$DSMASS, LENGT, WIDTH, HEIGH > 0$$

RELATION **DIESDAT**

Application: tabular description of the curves that limit the allowable region of the field of diesel engine performances described in relation DIESSFC. For gas turbines see relation TURBDAT.

ENGNO	OPCOND	RPM	DBR	MCR
---	---	---		
I07	A07	F06.01	F08.01	F08.01
IDENTIFIER	IDENTIFIER	ROT_SPEED	POWER	POWER

ENGNO: engine identifier in the DB. This attribute is the key of relation ENGINES and part of the key in relation DIESSFC.

OPCOND: defines the operating conditions such as sea- water and air temperatures. This attribute is the key in relation OPERCON where the corresponding operating conditions are described.

RPM: rotational speed in rpm.

DBR: maximum allowable intermediate power, kW, known also as *overload power*, or *power reserved for acceleration*. The values listed for this attribute may be used for limited time durations. The name of the attribute is an acronym for the German *Drehmoment-Begrenzungs Regler* which means "torque limiting controller". The line described by the sequence of ordered pairs (RPM, DBR) is actually the line of the maximum allowable torque.

MCR: maximum allowable continuous power, kW. May be used for unlimited time durations.

Default values:

OPCOND = STD

where this value is defined in relation DIESSFC.

Constraints:

$$RPM \geq 0$$

$$DBR \geq MCR \geq 0$$

RELATION **DIESSFC**

Application: describes the field of characteristic curves of diesel engine perfomances. The limiting curves of the field are stored in relation DIESDAT. The parallel relation for specific fuel consumption of gas turbines is TURBSFC.

ENGNO	OPCOND	RPM	POWER	SFC
---	---	---	---	
I07	A07	F06.01	F08.01	F05.03
IDENTIFIER	IDENTIFIER	ROT_SPEED	POWER	SPEC_FUEL_CONS

ENGNO: engine identifier in the DB. This attribute is the key of the relation ENGINE and part of the key in relation ENGDAT.

OPCOND: defines the operating conditions such as sea-water and air temperature. This attribute is the key of relation OPERCON where the corresponding operating conditions are described.

RPM: rotational speed in rpm. This attribute represents the abscissa of the characteristic curves of engine performance.

POWER: engine power in kW. This attribute represents the ordinate of the characteristic curves of engine performance.

SFC: specific fuel consumption in kg/kW.h (SI kg mass). This attribute represents the parameter of the characteristic curves of engine perfomance.

Default values:

$$OPCOND = STD$$

meaning the standard testing conditions defined by ISO 3046/I, respectively DIN 6271, that is

27°	ambient temperature
1000 mbar	atmospheric pressure
60%	rel. air humidity

27^{o} cooling water temperature before intercooler

Constraints:

$$RPM, POWER \geq 0$$

$$0 < SFC \leq 0.20$$

RELATION **PROPSER**

Application: defines the main characteristics of propellers the data of which are stored in the DB. For hydrodynamic performances see relation PROPDAT, and for constructive data of actually existing propellers see relation PROPPAR.

PROPNO	PMAKE	PTYPE	Z
I06	A15	A15	I02
IDENTIFIER	IDENTIFIER	IDENTIFIER	NONDIM

EAR	BLADE	PT	CAV
F05.03	A10	A03	A02
NONDIM	TEXT	TEXT	TEXT

PROPNO: propeller identifier within the DB. This attribute constitutes a surrogate key, that is a key assigned sequentially by the system. PROPNO is part of the key in relation PROPDAT and key in relation PROPPAR. We can distinguish the following three cases:

1) PROPNO is associated with one fixed pitch propeller (PT = FPP or PT = VPP in relation PROPSER);

2) PROPNO is assigned to a series of fixed pitch propellers (PT = FPP or PT = VPP in relation PROPSER), defined by one single tuple in relation PROPSER. Then, the same PROPNO corresponds to a sequence of POD values, each POD value representing a different propeller subseries;

3) PROPNO designs a controllable pitch propeller (PT = CPP or PT = CRP in relation PROPSER). Then, POD can assume a sequence of different values for the same popeller, at one and the same PROPNO value.

PMAKE: identifier of either the propeller series or the producer, e. g. B-SERIES, Gawn-Burrill, Newton-Rader, KaMeWa.

PTYPE: propeller identifier within PMAKE, e. g. B85-10.

Z: number of blades.

EAR: expanded-area ratio.

BLADE: blade section, e. g. NACA 0.8.

PT: pitch characterization, e. g.

CPP = controllable pitch. The meaning is "controllable as a function of time".

CRP = controllable reversible pitch (see note above).

FPP = fixed pitch. The meaning is "fixed in time".

VPP = variable pitch. The meaning is "variable as function of radius, but fixed in time".

CAV: propeller regime characterization, e. g.

CA = cavitating

SC = supercavitating

SP = surface piercing

TC = transcavitating

UC = undercavitating

Default values:

Z = 3

PT = VPP

CAV = UC

Constraints:

$$2 \leq z \leq 7$$

RELATION **PROPPAR** (propeller particulars)

Application:

stores the particulars of propellers existing in owner's inventory. Thus, PROPPAR is essentially a propeller catalog of one given owner.

PROPNO	PROPVA	DIA	POD	PLOC

I06	I03	F05.03	F05.03	A05
IDENTIFIER	IDENTIFIER	LENGTH	NONDIM	TEXT

ROT	DHUB	PMAS	PINERT	CATNO
A02	F05.03	F05.03	F06.04	A09
TEXT	LENGTH	MASS	INERTIA	TEXT

PROPNO: propeller identifier within the DB, defined in relation PROPSER. This attribute constitutes a surrogate key, that is a key assigned sequentially by the system. PROPNO may be the identifier of a series of propellers the performances of which are presented in nondimensional form.

PROPVA: identifies a constructive propeller within the series PROPNO. That is, while PROPNO defines a certain series with given nondimensional Z and EAR characteristics (see relation PROPSER), PROPVA identifies a propeller with a given dimensional diameter DIA.

DIA: propeller diameter in m.

POD: pitch-to-diameter ratio, P/D . If propeller performances are known and stored in relation PROPDAT, POD should be the same as for the identical PROPNO in relation PROPDAT.

If PT in relation PROPSER equals either CPP or CRP, POD may assume an infinite number of values within a certain range. Then, the maximum POD value should be stored here.

PLOC: defines the way in which POD is measured. For example:

AVERG means that POD is the mean weighted P/D ratio over the propeller radius;

0.7R means that P/D is measured at $0.7DIA/2$.

ROT: defines the sense of rotation, when looking forward:

ROT = RH when right-handed;

ROT = LH when left-handed.

DHUB: hub diameter, in m.

PMAS: propeller mass in t (SI ton mass).

PINERT: propeller mass moment of inertia about its axis in tm^2 (SI unit).

CATNO: owner's catalog number.

RELATION **PROPDAT**

Application:

- lists open-water propeller performances.

PROPNO	POD	SIGMA	J	KT	KQ
I06	F05.03	F06.03	F06.03	F06.04	F07.05
IDENTIFIER	NONDIM	NONDIM	NONDIM	NONDIM	NONDIM

PROPNO: propeller identifier within the DB, defined in relation PROPSER. See comments in relation PROPSER.

POD: pitch-to-diameter ratio, P/D. If this is an actually existing propeller the constructive data of which are stored in relation PROPPAR, then POD must be the same in both relations, for the same PROPNO.

SIGMA: cavitation number (σ) defined as

$$\Sigma = \frac{p_a + \rho g d - p_v}{1/2 \rho V_R^2}$$

where

p_a is the atmospheric pressure

ρ is water density at prevailing temperature

g is gravity acceleration

d is the draft of the propeller axis

p_v is vapour pressure at prevailing temperature

V_R is the relative velocity at the representative radius R.

Care must be taken to calculate the cavitation number under exactly the same conditions that were used by the cavitation tunnel that published the data stored in this relation.

J: advance coefficient defined as

$$J = \frac{V_A}{nD}$$

where V_A is the *speed of advance*, n is the rotational speed and D the diameter of the propeller.

KT: thrust coefficient defined as

$$K_T = \frac{T}{\rho n^2 D^4}$$

where T is propeller thrust and ρ water density.

KQ: torque coefficient defined as

$$K_Q = \frac{Q}{\rho n^2 D^5}$$

where Q is the torque applied on propeller shaft.

Default values:

SIGMA = atmospheric

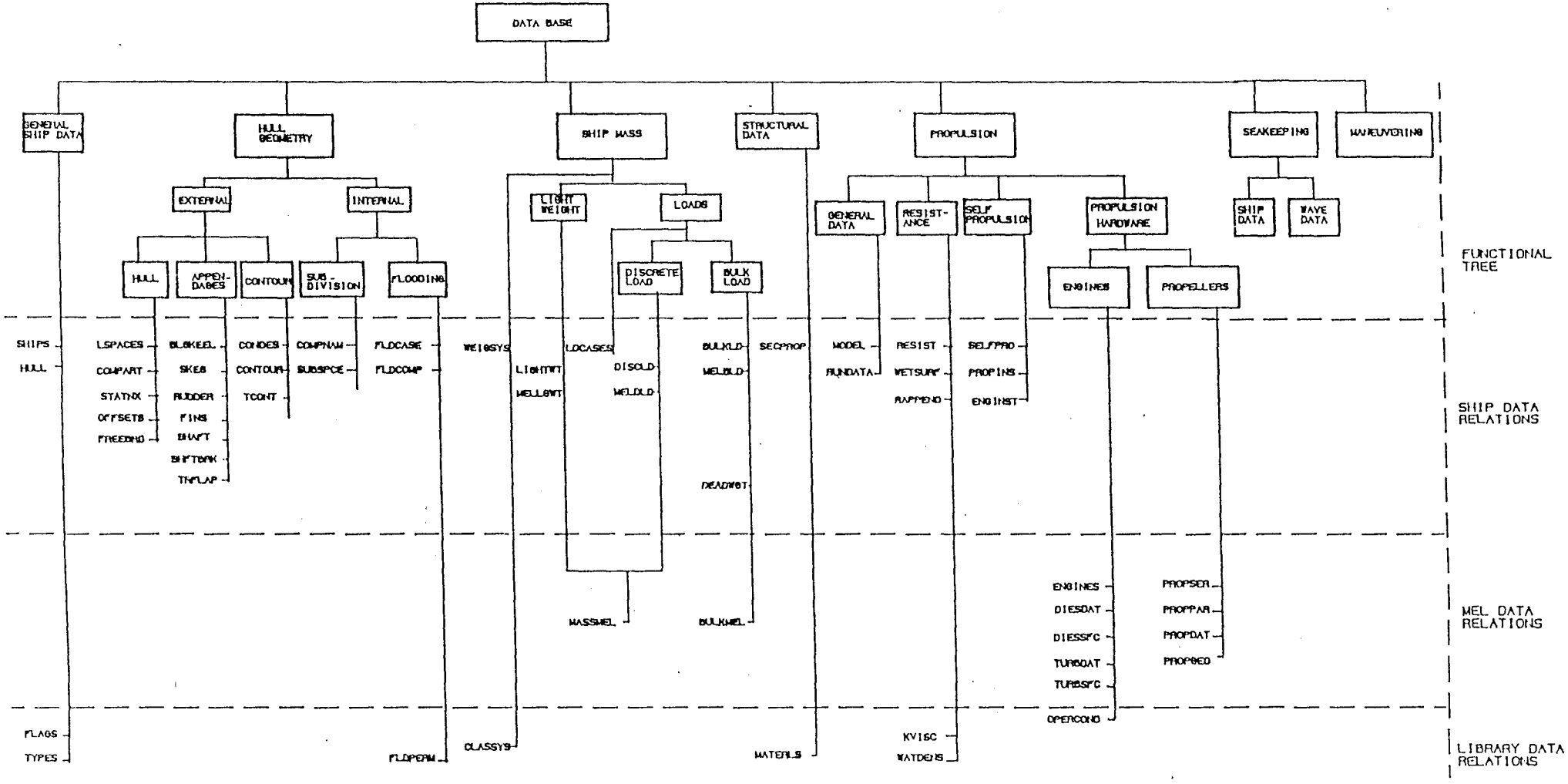
Note:

the open-water efficiency η_o is not stored in the DB as it can be derived from

$$\eta_o = \frac{J}{2\pi} \cdot \frac{K_T}{K_Q}$$

List of figures

FIG 1: THE CONTENTS OF THE DATABASE



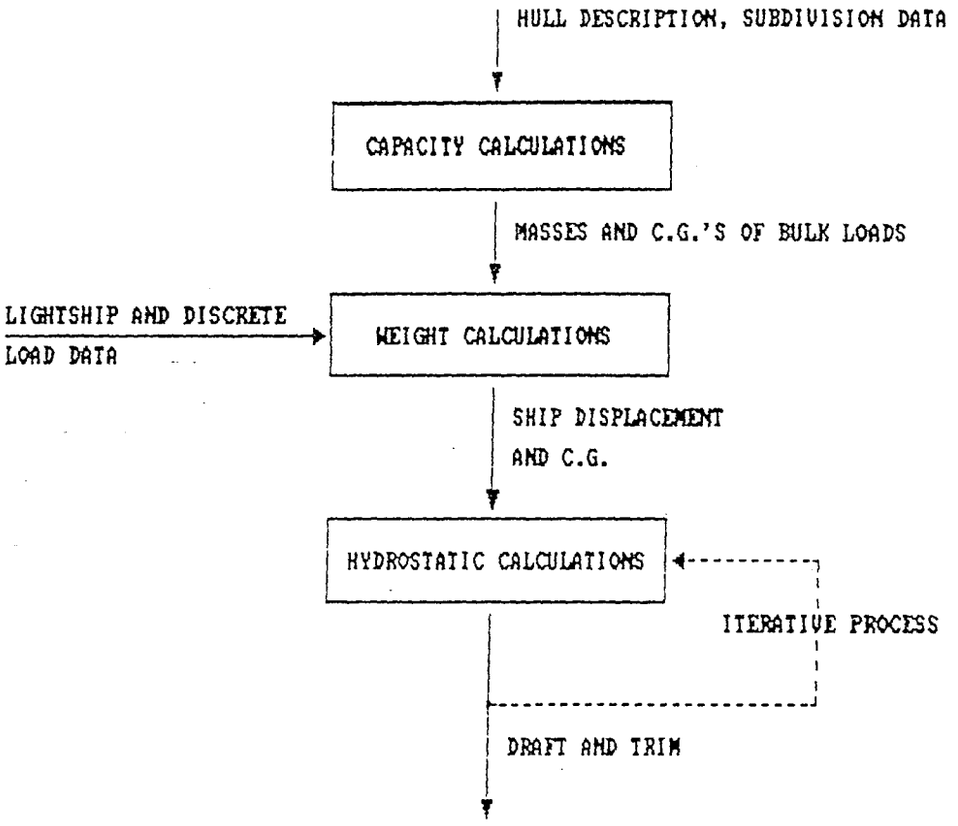

HULL DESCRIPTION, SUBDIVISION DATA
CAPACITY CALCULATIONS
MASSES AND C.G.'S OF BULK LOADS
LIGHTSHIP AND DISCRETE
LOAD DATA
WEIGHT CALCULATIONS
SHIP DISPLACEMENT
AND C.G.
HYDROSTATIC CALCULATIONS
ITERATIVE PROCESS
DRAFT AND TRIM

Fig. 2

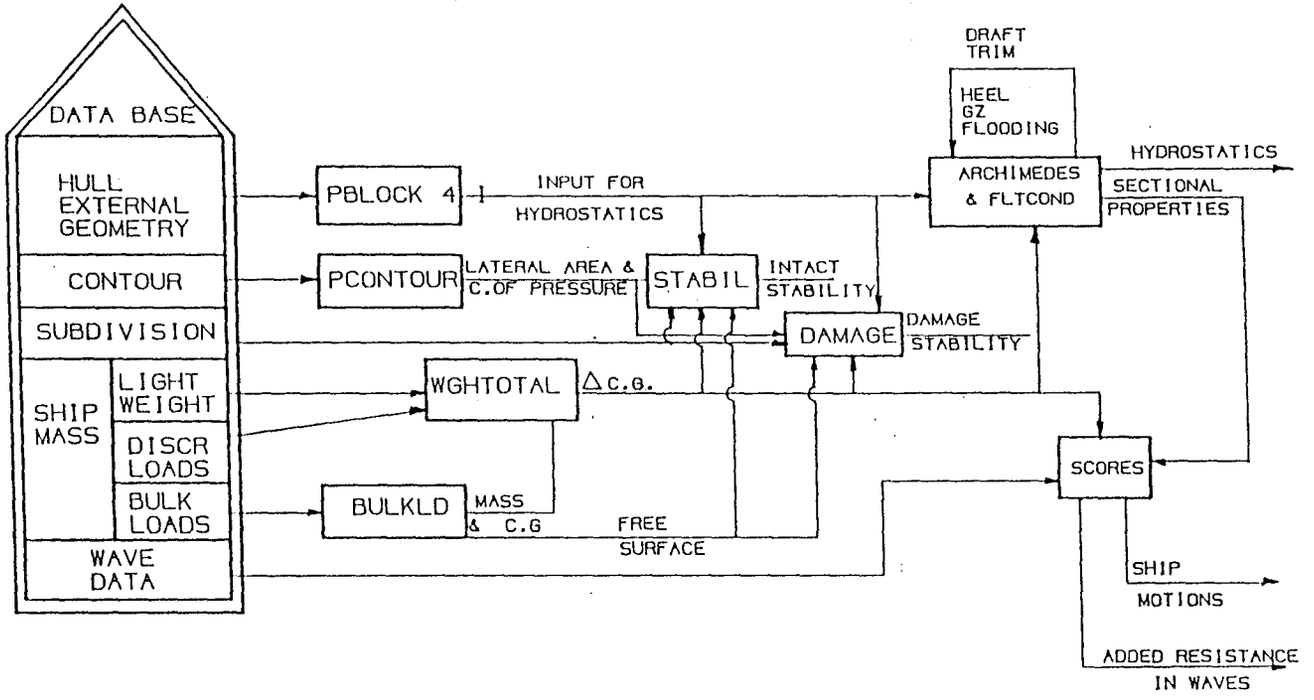


FIG 3-DATA FLOW AND CALCULATION RUNS OF A CONCENTRATED MASS SHIP MODEL.

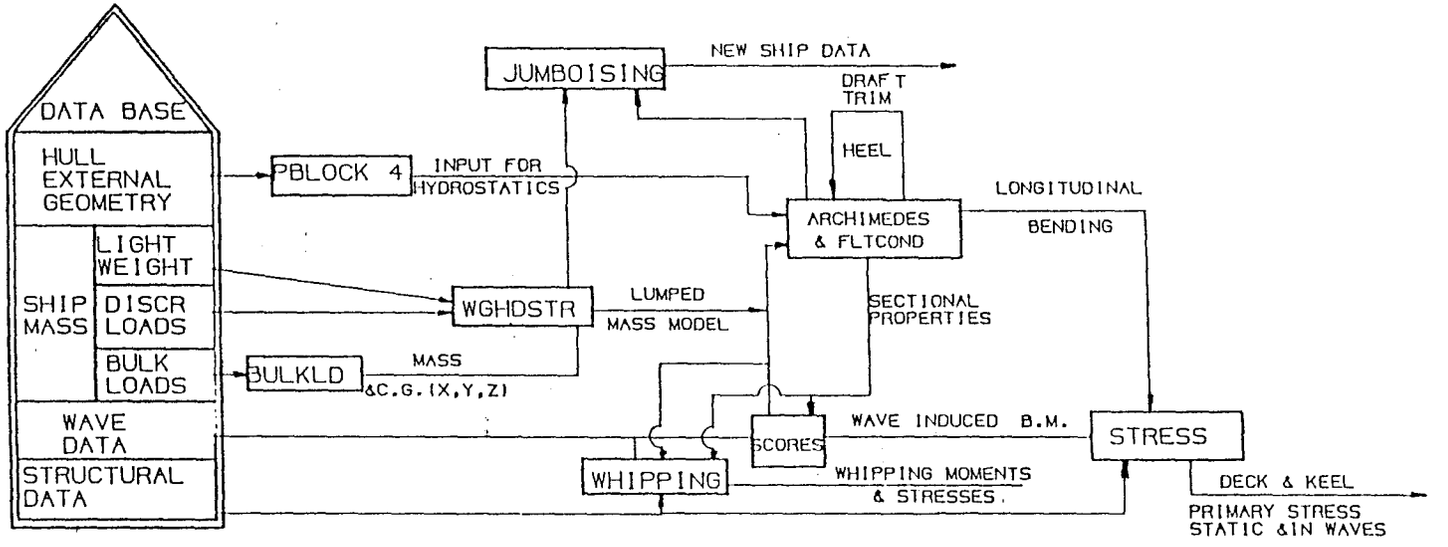


FIG 4-DATA FLOW AND CALCULATION RUNS OF A LUMPED MASS SHIP MODEL.

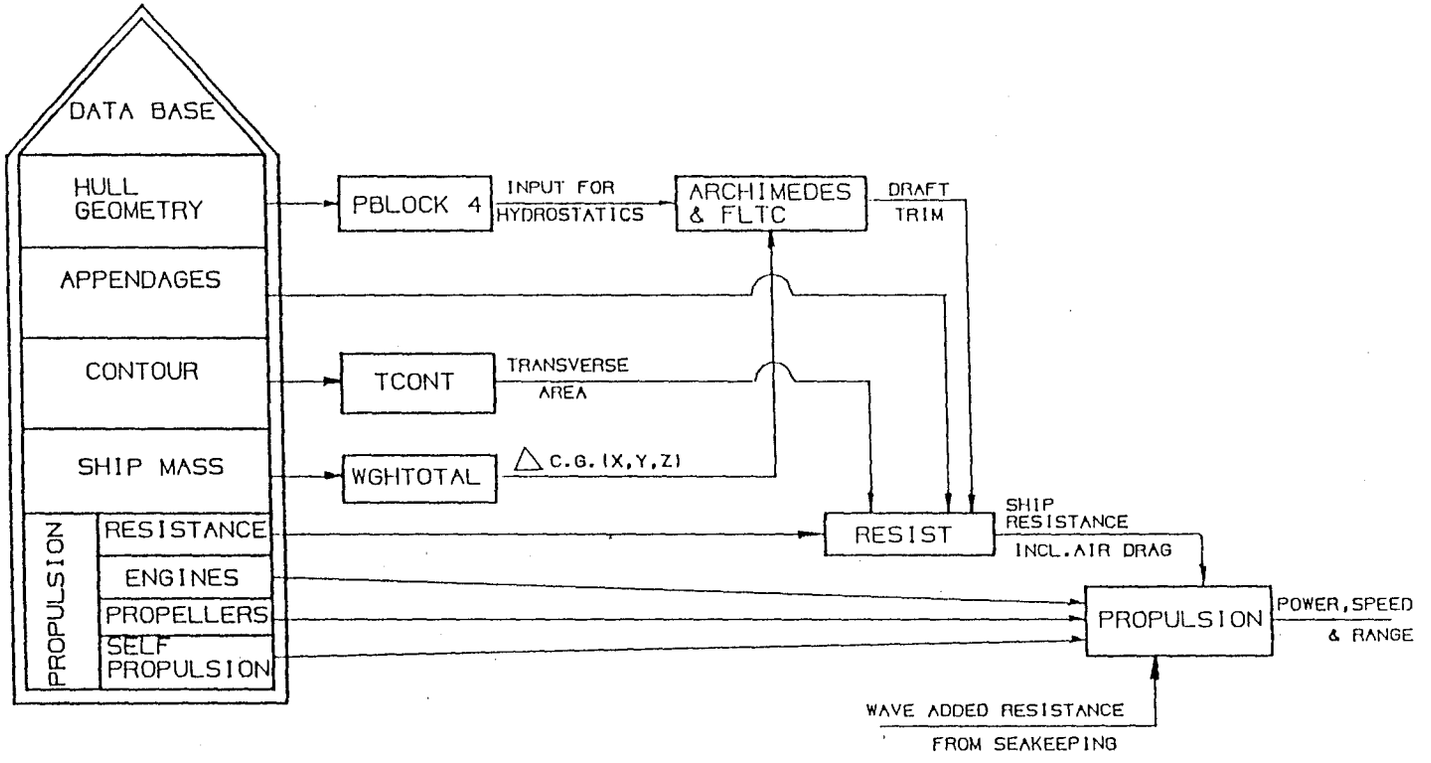
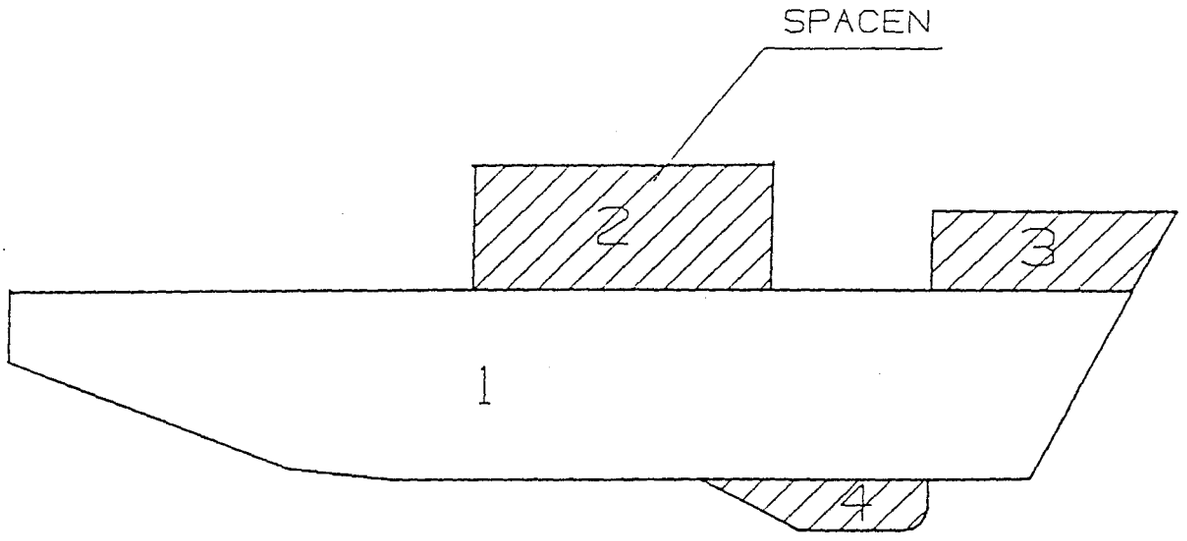


FIG 5- DATA FLOW AND CALCULATION RUNS FOR POWERING AND PROPULSION



NOTE: SPACEN=0 INCLUDES THE WHOLE SHIP (1+2+3+4)

FIG 6: TO RELATION LSPACES

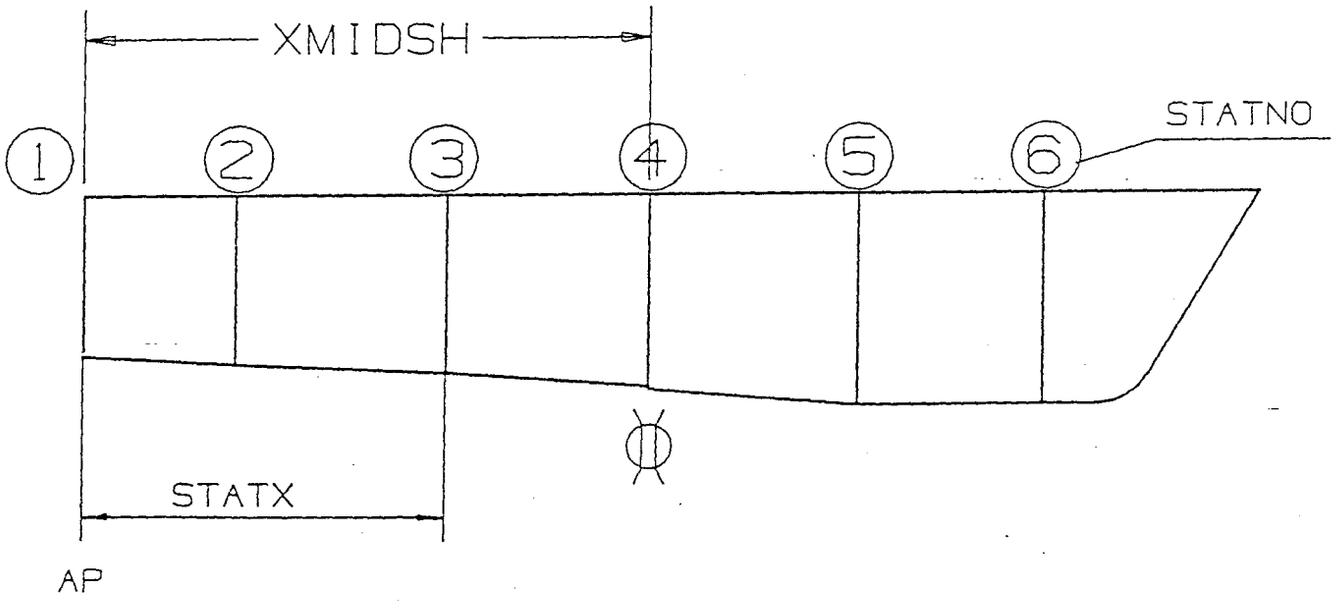


FIG 7: TO RELATION STATNX

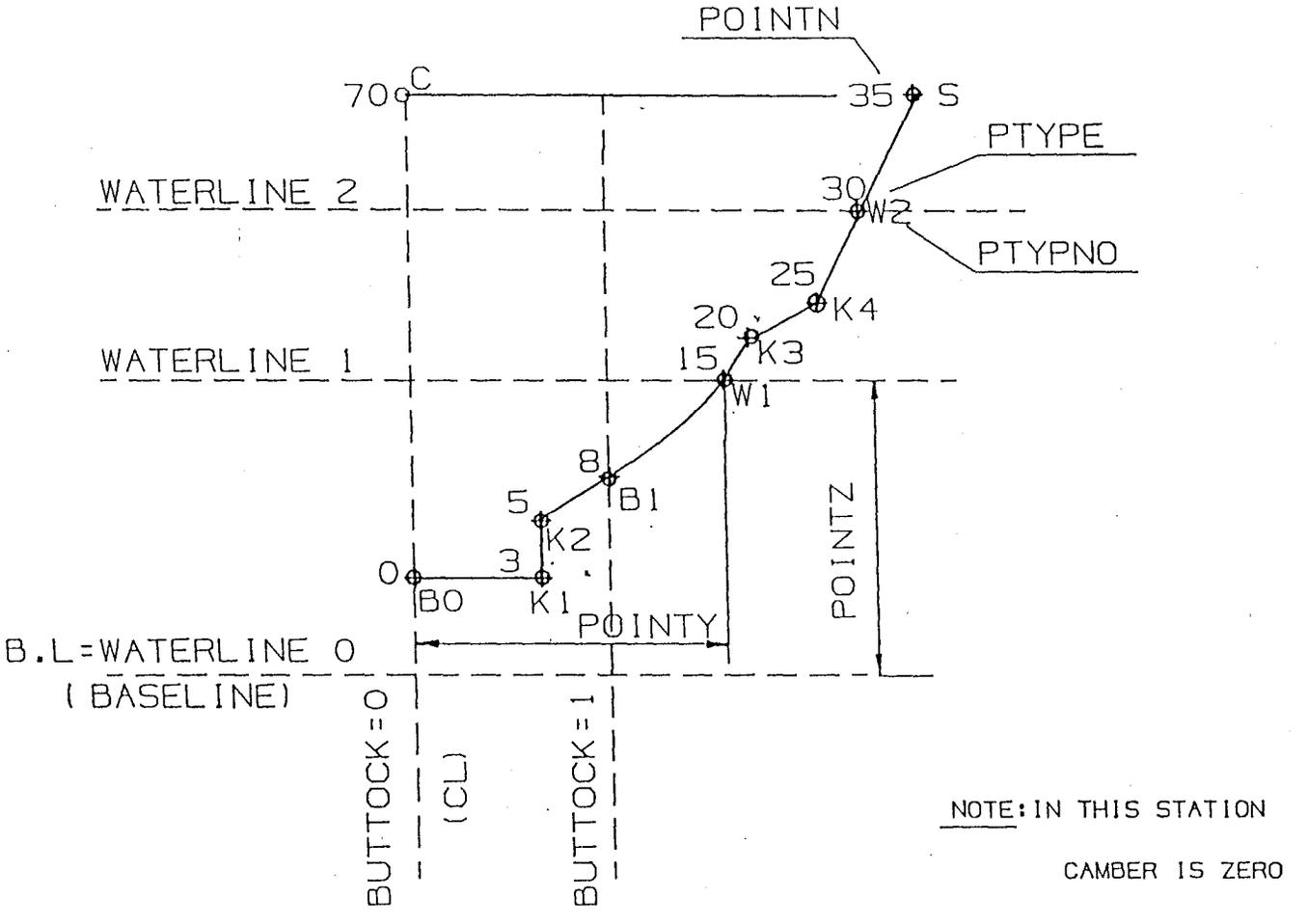


FIG 8: TO RELATION OFFSETS

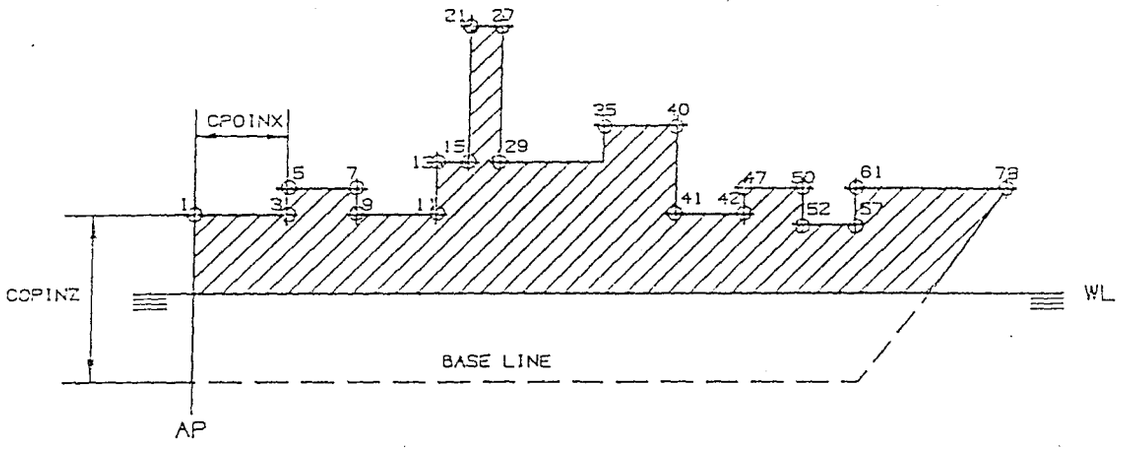


FIG 9:TO RELATION CONTOUR

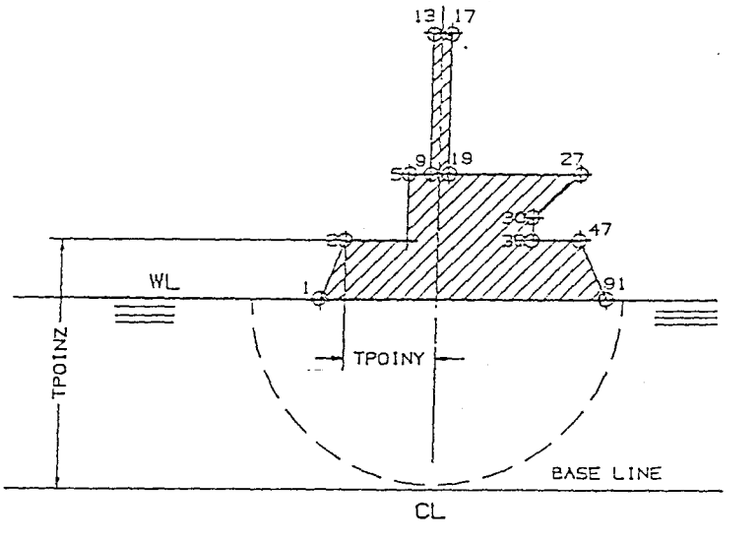
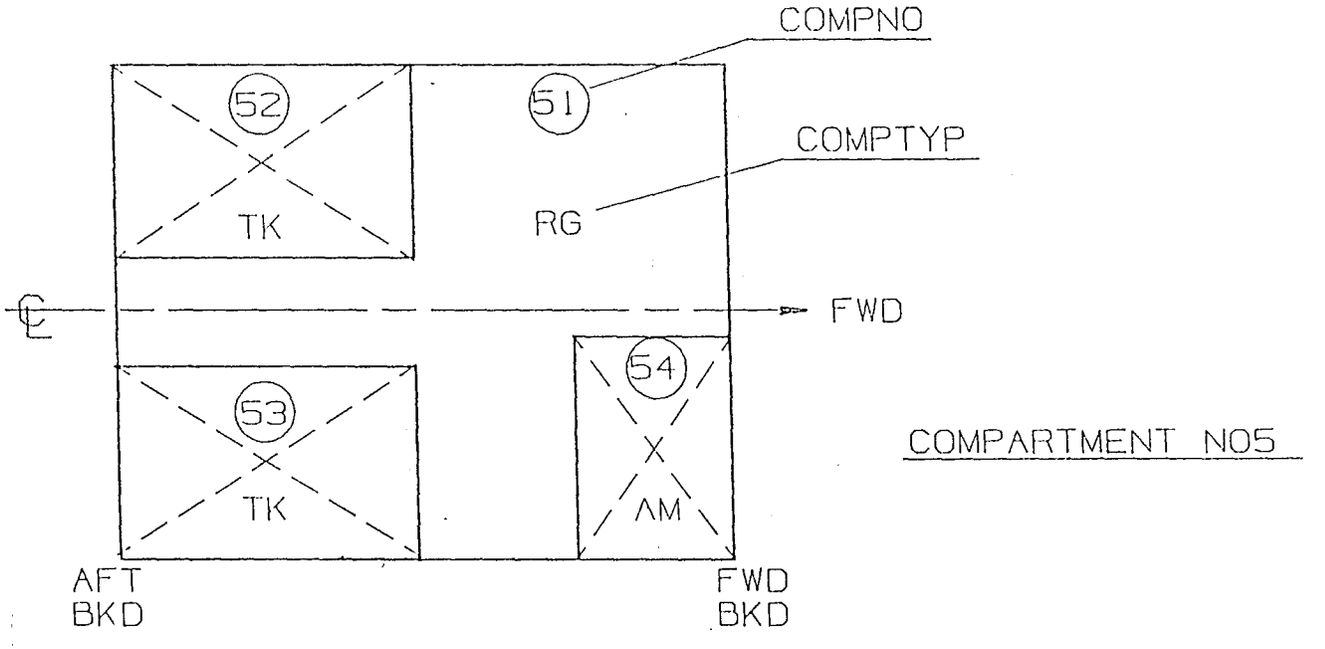


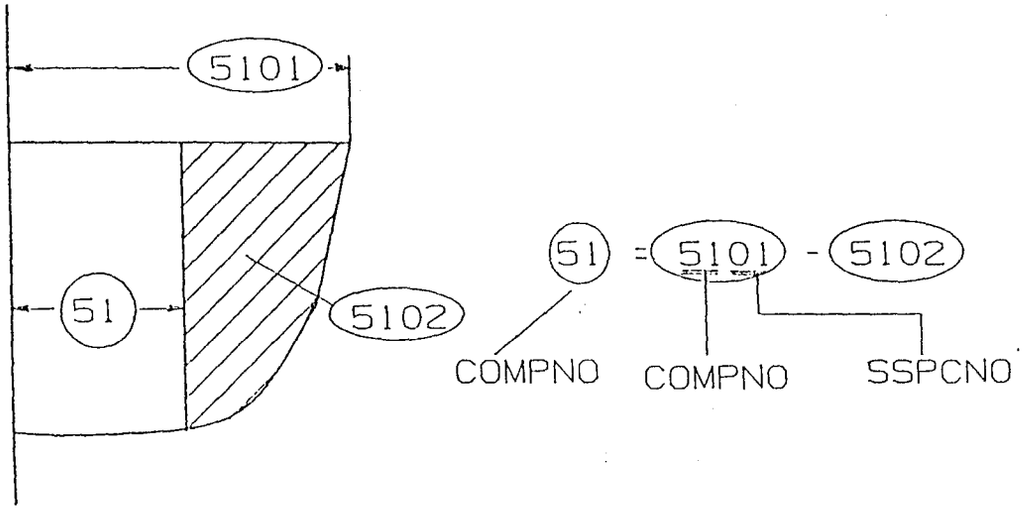
FIG 10: TO RELATION TCONT



COMPARTMENT NO5

FIG 11: PLANVIEW OF A TYPICAL COMPARTMENT

(TO RELATION COMPNAM)



SECTION VIEW OF COMPNO 51

FIG 12: TO RELATION SUBSPCE

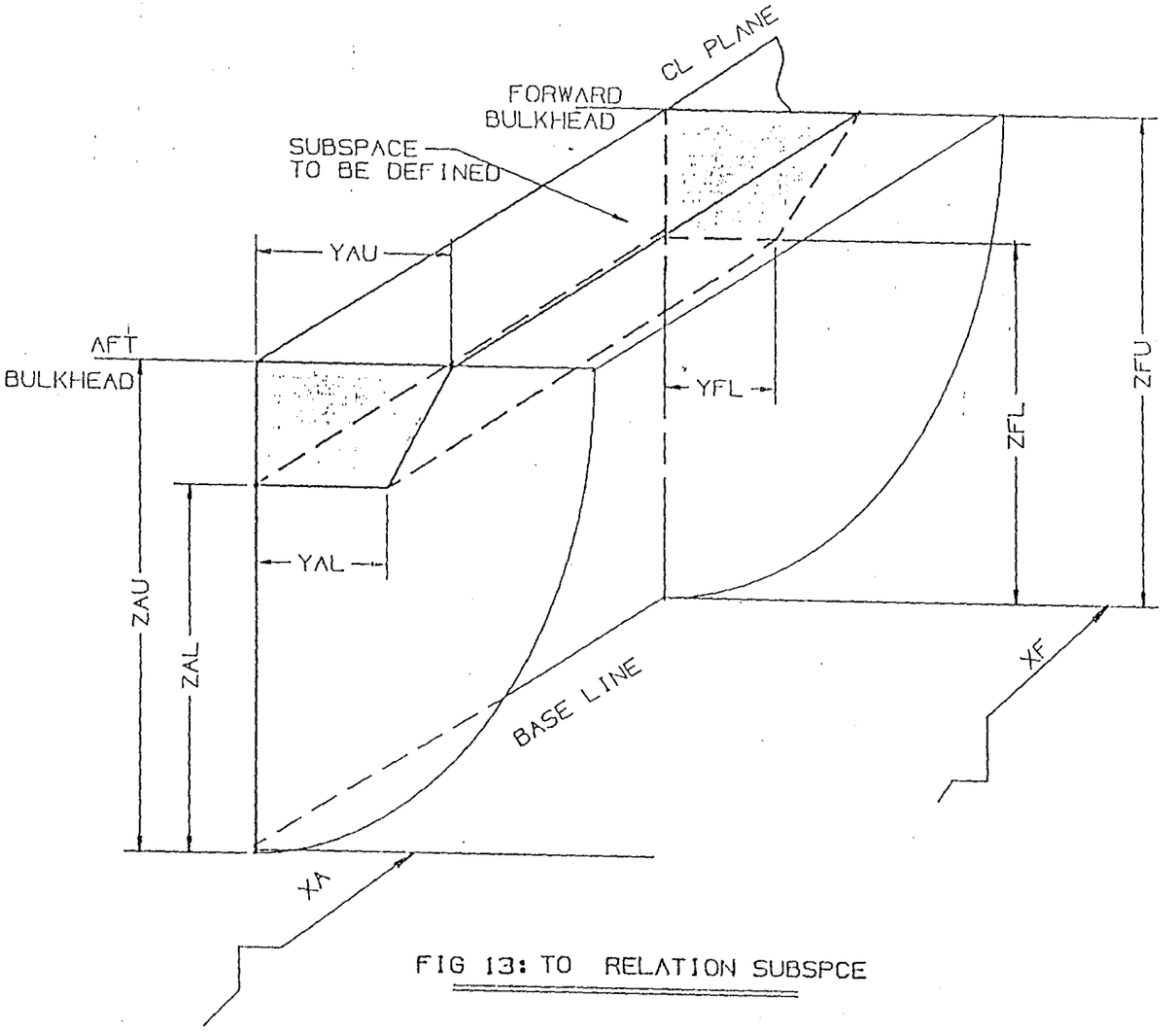


FIG 13: TO RELATION SUBSPCE