University of L'Aquila

Department of Information Engineering, Computer Science and Mathematics

NavibusMercatoriis:
A Database for a Mercantile Shipping Company

Databases - Lab Module - Final Project

Student
Name :
Aly Shmahell

Supervisor
Name :
Prof. Pierluigi Pierini

Jan 9th 2018

Abstract

As part of a lab course on databases, I, the student, was tasked with designing a database for a mercantile shipping company, starting from specifications given by the supervisor, according to industry standards. In this document I showcase the four stages of design, along with the design decisions that are made beforehand. Great care is given into finding optimal solutions, but in some extreme cases a local optima is chosen for conformity purposes (the solution conforms best to the design stage), for practicality or for highlighting the optimal solution.

https://github.com/AlyShmahell/NavibusMercatoriis

Design Decisions

Table 1: Design Tool	
Question	Which design tool to choose?
Options	• Dia • MySQL Workbench
Criteria	• being up to date • offering ER models
Evaluation	• MySQL Workbench satisfies both criteria, it also comes equipped with SQL scripting, python scripting, SQL database connection, and an SQL linter/debugger. • Dia is outdated and lacks features.
Choice	MySQL Workbench

Table 2: Testing Tool	
Question	Which testing tool to choose?
Options	• phpmyadmin • adminer
Criteria	• being up to date. • offering PL/SQL support.
Evaluation	• adminer satisfies both criteria. • phpmyadmin lacks proper support for delimiters in stored procedures.
Choice	adminer

Stage 1: Conceptual Design

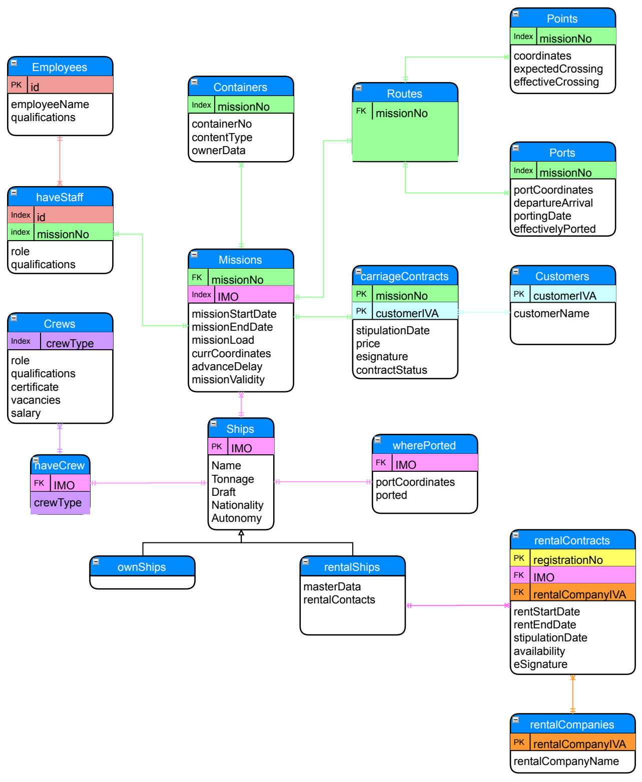
For the Conceptual Schema I used Crow's Foot notation, mainly because it is the one adopted by the CASE tool I chose (MySQL WorkBench).

With this notation, relationships cannot have attributes. When necessary, relationships are promoted to entities. However, when relationships are not promoted to entities, they represent external identifiers as foreign key or index attributes which become present in both entities on the left and right hand side.

This results in the following:

- adaptation of external identifiers into foreign keys early on in the conceptual design stage.
- selection of primary identifiers can also be done in the conceptual design stage.
- many-to-many relationships are replaced with entities in the conceptual design stage.

Figure 1: Conceptual ERD with Crow's Foot (IE) Notation



Stage 2: Logical Design

Data Dictionary - Entities

Entity	**Description**	**Attributes**	**Identifier**
Ships	Represents all Ships.	IMO, name, tonnage, nationality, draft, autonomy	IMO
ownShips	Represents Company Ships.	IMO	IMO
rentalShips	Represents Ship Rented from Rental Companies.	IMO, masterData, rentalCompanyContacts	IMO
rentalCompanies	Companies we rent ships from	rentalCompanyIVA, rentalCompanyName	rentalCompanyIVA
rentalContracts	Contracts between our company and the companies we rent ships from.	registrationNo, rentalCompanyIVA, IMO, rentStartDate, rentEndDate, availability, esignature	registrationNo
haveCrew	Represents associations between certain ships and certain crew types.	IMO, crewType	IMO
Crews	represent the various roles of the various crew types.	Pk, crewType, role, qualifications, certificate, vacancies, salary	pk
wherePorted	represents the current port of each ship in the present, independent of mission ports.	IMO, portCoordinates, ported	IMO
Missions	represents the missions corresponding to carriage contracts.	missionNo, IMO, missionStartDate, missionEndDate, missionLoad, advanceDelay, currCoordinates, missionValidity	missionNo
Employees	represents all the employees of the company, regardless of their current specific occupation at the company.	id, employeeName, qualifications	id
haveStaff	represents the staff of each mission, who are picked from the employees according to crew type.	missionNo, id, role, qualifications.	missionNo
Customers	represents the customers who want to request carriage missions.	customerIVa, customerName	customerIVA
carriageContracts	carriage contracts between our company and a customer.	missionNo, customerIVA, stipulationDate, price, esignature, contractStatus	missionNo
Containers	describes each individual container of each mission.	missionNo, containerNo, contentType, ownerData	missionNo
routes	identifies the route of each mission	missionNo	missionNo
Points	represents the geographic points of each route.	missionNo, coordinates, expectedCrossing, effectiveCrossing	missionNo
Ports	represents ports associated with missions.	missionNo, portCoordinates, departureArrival, portingDate, effectivelyPorted	missionNo

Business Rules – Constraints

BR1	Every rentalShips or ownShips entry must have a Ships entry to associate it with.
BR2	Every rentalContracts Entry must have a rentalShip and a rentalCompany added to their tables respectively.
BR3	Every haveCrew entry must have a Ship to associate it with.
BR4	Every wherePorted entry must have a Ship to associate it with.
BR5	Every Crews entry must have a haveCrew entry to associate it with.
BR6	Every carriageContracts entry must have a Customers entry to associate it with.
BR7	Every Missions entry must have both a carriageContracts entry and a Ships entry to associate it with.
BR8	Every haveStaff must only be added after a proper Ships entry has been assigned to the Missions entry.
BR9	Every haveStaff entry must only be added if the staff is available in the Missions entry date parameters.
BR10	Every Ships entry associated with the Missions entry must satisfy availability within the Missions entry date Parameters.
BR11	Every Ships entry associated with the Missions entry must satisfy same wherePorted or Ports entry in repect to the Missions entry date parameters.
BR12	Every haveStaff entry should satisfy the Qualifications of the Crews entry associated via haveCrew entry to the Ships entry which is in turn associated with the Missions entry.
BR13	Every haveStaff entry must correspond to a Employees entry already present.
BR14	Every containers entry must correspond to a Missions entry already present.
BR15	Every Routes entry must correspond to a Missions entry already present.
BR16	Every Points entry must correspond to a Routes entry already present.
BR17	Every Ports entry must correspond to a Routes entry already present.

Business Rules – Derivations

BR18	calculateTurnOver of the last 30 days is obtained by summing price cell of the Missions entries that have stipulationDate cell which has less than 30 days difference from now.
BR19	detectCollision is obtained by comparing likeliness in coordinates cell in Points entries corresponding to Routes entries corresponding to Missions entries that have intersecting startDate and endDate cell values.
BR20	missionLoad is obtained by summing the number of Containers entries associated with the Missions entry.
BR21	missionValidity is obtained by comparing missionLoad of the Missions entry to the tonnage of the associated Ships entry
BR22	checkPersonnelNo is obtained by summing the number of haveStaff entries corresponding to the Missions entry.

Table of Volumes (after 1 Year)

Table	Size Range	Notes
Ships	30 – 100	30 total ships on average, amped up to 100 to fill a 100 missions in a worst case scenario.
ownShips	10 – 10	
rentalShips	20 – 90	20 rental ships on average, amped up to 90 to fill a 100 missions in a worst case scenario.
rentalCompanies	1 – 90	
rentalContracts	20 – 90	
wherePorted	30 – 100	
haveCrew	30 – 100	
Crews	25 – 2500	100 missions, 25 crew roles (2500 in a worst case scenario).
Customers	1 – 100	
Missions	100 – 100	
carriageContracts	100 – 100	
Containers	300000 – 600000	
Employees	25 – 2500	
haveStaff	2500 – 2500	each mission has 25 unique staff entries, 100 missions have 2500.
routes	100 – 100	
Points	100 * x – 100 * y	100 missions per year, unknown number of points.
Ports	200 – 200	100 missions per year, 2 ports each.

Table of Routines

Procedure/Trigger	Frequency (per Year)	Notes
calculateTurnover	12	12 months per year
chooseProperShip	100	100 missions per year
pickStaff	100	
detectCollision	100	
checkPersonnelNo	100	
setMissionValidity	100	
triggerEffectivePorting	200	100 missions per year, 2 ports each.
triggerEffectiveCrossing	100 * x – 100 * y	100 missions per year, unknown number of points.
calculateMissionLoad	100	100 missions per year
reCalculateMissionLoad	297000 – 594000	100 missions per year, 3000 – 6000 containers each.
checkOwnShipsNumber	0 – 10	10 company ships
checkRentalShipsNumber	0 – 90	
addStipulationDate	0 – 90	
updateStipulationDate	0 – 90	
addRentalShipAvailability	0 – 90	
updateRentalShipAvailability	0 – 90	

Tables of Accesses

Table	**Accesses (per Day)**	**Type**
Ships	0 – 360	R
Missions	0 – 120	R
ownShips	0 – 330	R
rentalShips	0 – 30	R
Containers	90000 – 180000	W
haveStaff	1800	R
haveStaff	0 – 750	W
Routes	0 – 120	R
Points	0 – 120 * x	R
Points	0 – 30 * x	W
Ports	0 – 150	R
Ports	0 – 60	W
Employees	0 – 60	R
Routes	0	R/W
haveCrew	0 – 60	R
Crews	30	R
wherePorted	30	R

Note: each of these entries were calculated by taking into consideration that all entries could be accessed by the procedures in a day (for example, all 30/30 ships on average) then multiplying that number by the number of accesses (Select in case of Read, Insert/Update in case of Write).

Stage 3: Normalization

Removal of a generalizations was done following the **"Substitution of the generalization with relationships"** method.

The resulting translated tables were as the following:

Ships(IMO, name, tonnage, draft, autonomy, nationality)
ownShips(IMO)
rentalShips(IMO, masterData, rentalContacts)
rentalCompanies(rentalCompanyIVA,rentalCompanyName)
rentalContracts(registrationNo, rentalCompanyIVA, IMO, rentStartDate, rentEndDate, availability, eSignature)
wherePorted(IMO, portCoordinates, ported)
haveCrew(IMO, crewType)
Crews(*crewType*, role, qualifications, certificate, vacancies, salary)[1]
Missions(missionNo, IMO, missionStartDate, missionEndDate, missionLoad, currCoordinates, advanceDelay, missionValidity)
Customers(customerIVA, customerName)
Employees(id, employeeName, qualifications)
carriageContracts(missionNo, customerIVA, stipulationDate, price, eSignature, contractStatus)
haveStaff (*id*, *missionNo*, role, qualifications)[1]
Containers(*missionNo*, containerNo, contentType, ownerData)[1]
Routes(missionNo)
Ports(*missionNo*, portCoordinates, departureArrival, portingDate, effectivelyPorted)[1]
Points(*missionNo*, coordinates, expectedCrossing, effectiveCrossing)[1]

The resulting translated foreign key relations are as the following:

`fk_rentalContracts_rentalCompanies1`(rentalContracts.rentalCompanyIVA, rentalCompanies.rentalCompanyIVA)
`fk_rentalContracts_rentalShips1`(rentalContracts.IMO, rentalShips.IMO)
`fk_rentalShips_Ships1`(rentalShips.IMO,Ships.IMO)
`fk_ownShips_Ships1`(ownShips.IMO, Ships.IMO)
`fk_wherePorted_Ships1`(wherePorted.IMO, Ships.IMO)
`fk_haveCrew_Ships1`(haveCrew.IMO, Ships.IMO)
`fk_Crews_haveCrew1`(Crews.crewType, haveCrews.crewType)
`fk_Missions_Ships1`(Missions.IMO, Ships.IMO)
`fk_carriageContract_Customer1`(carriageContracts.customerIVA, Customers.customerIVA)
`fk_Missions_carriageContracts1`(Missions.MissionNo, carriageContracts.MissionNo)
`fk_haveStaff_Employees1`(haveStaff.id, Employees.id)
`fk_haveStaff_Missions1`(haveStaff.missionNo, Missions.missionNo)
`fk_Loads_Missions1`(Containers.missionNo, Missions.missionNo)
`fk_Routes_Missions1`(Routes.missionNo, Missions.missionNo)
`fk_Ports_Routes1`(Ports.missionNo, Routes.missionNo)
`fk_Points_Routes1`(Points.missionNo, Routes.missionNo)

Notes on Efficiency
As we can see in the table of accesses, the **Routes** table is accessed very rarely, and it imposes an un-necessary structural redundancy (only contains missionNo), therefor should be deleted and substituted with the **"missionNo"** attribute in the **Missions** table.
Also, the **ownShips** table is accessed as frequently as the **Ships** table, but poses another un-necessary structural redundancy, therefor should be deleted and substituted with a binary attribute **"isOwn"** in the **Ships** table.

[1] attributes in italic represent indexes

Figure 2: Relational EERD with Crow's Foot (IE) Notation

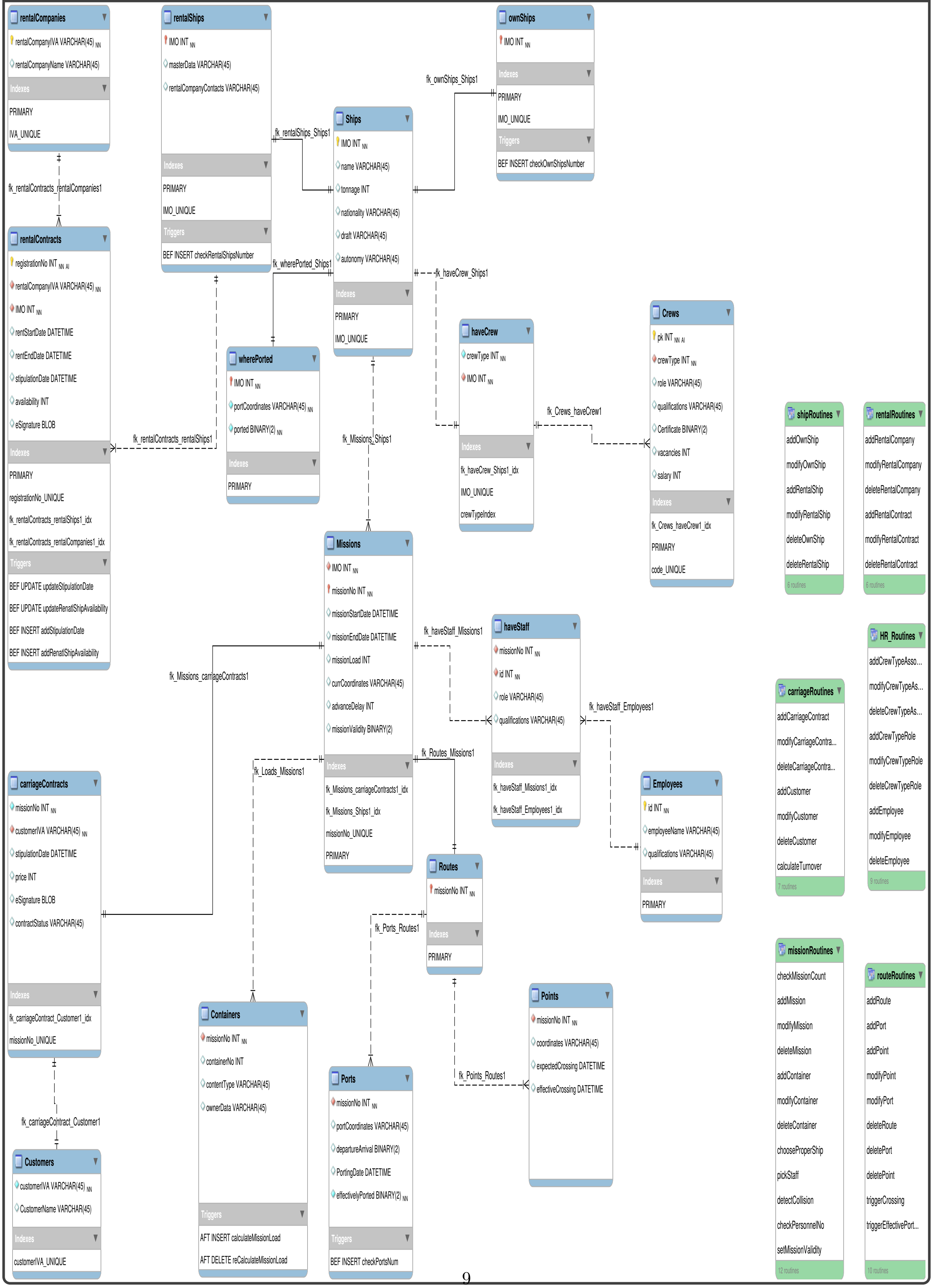
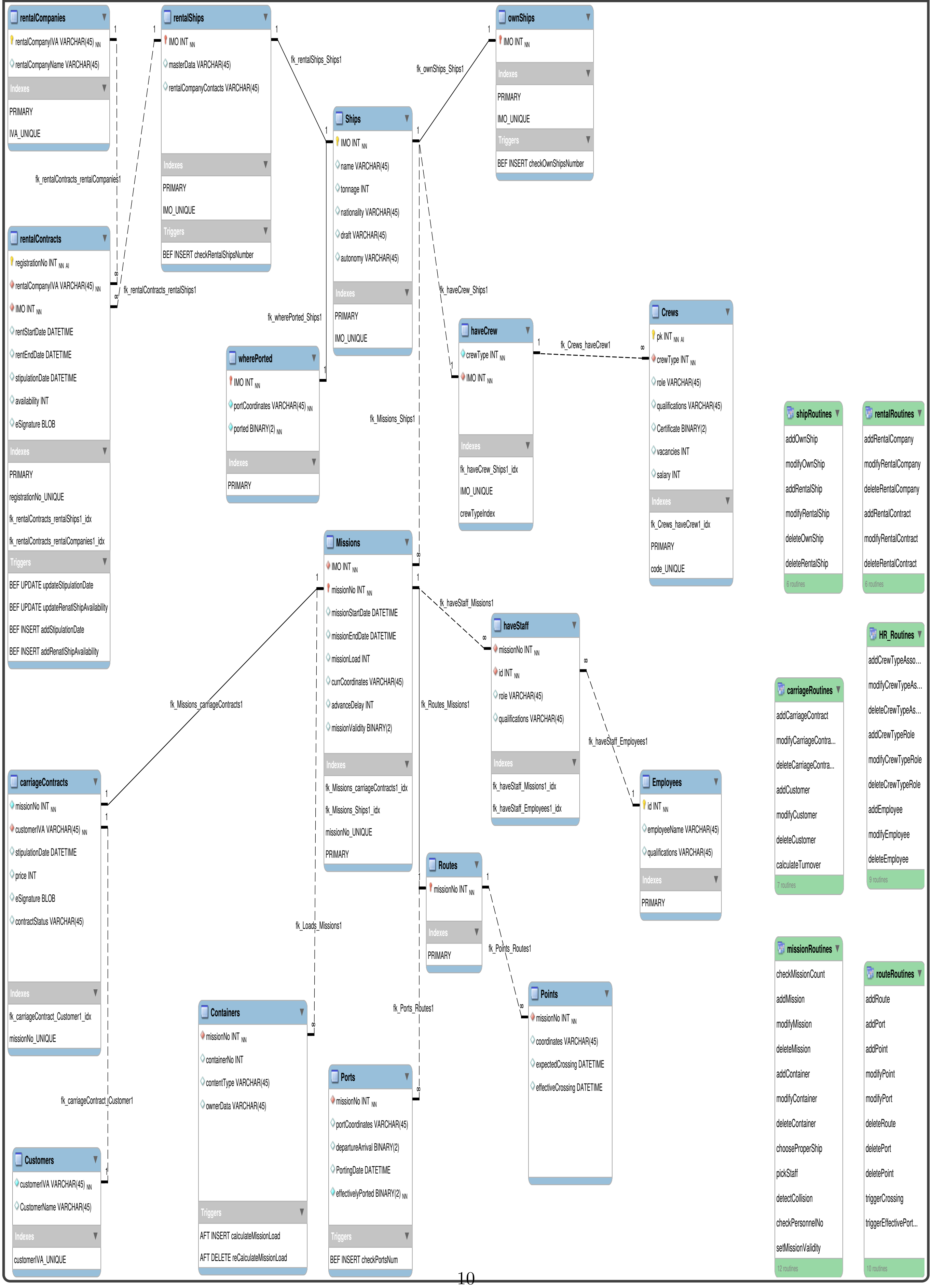


Figure 3: Logical EERD with Column Matching Notation



Stage 4: Physical Schema

```
-- MySQL Script generated by MySQL Workbench
-- dom 14 gen 2018 02:07:18 CET
-- Model: Mercantile Ships    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema Mercantile Ships
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema Mercantile Ships
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `Mercantile Ships` DEFAULT CHARACTER SET utf8 ;
USE `Mercantile Ships` ;

-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Ships`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Ships` (
  `IMO` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `tonnage` INT NULL,
  `nationality` VARCHAR(45) NULL,
  `draft` VARCHAR(45) NULL,
  `autonomy` VARCHAR(45) NULL,
  PRIMARY KEY (`IMO`),
  UNIQUE INDEX `IMO_UNIQUE` (`IMO` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`rentalCompanies`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`rentalCompanies` (
  `rentalCompanyIVA` VARCHAR(45) NOT NULL,
  `rentalCompanyName` VARCHAR(45) NULL,
  PRIMARY KEY (`rentalCompanyIVA`),
  UNIQUE INDEX `IVA_UNIQUE` (`rentalCompanyIVA` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`rentalShips`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`rentalShips` (
  `IMO` INT NOT NULL,
  `masterData` VARCHAR(45) NULL,
  `rentalCompanyContacts` VARCHAR(45) NULL,
  PRIMARY KEY (`IMO`),
  UNIQUE INDEX `IMO_UNIQUE` (`IMO` ASC),
  CONSTRAINT `fk_rentalShips_Ships1`
    FOREIGN KEY (`IMO`)
    REFERENCES `Mercantile Ships`.`Ships` (`IMO`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`ownShips`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`ownShips` (
  `IMO` INT NOT NULL,
  PRIMARY KEY (`IMO`),
  UNIQUE INDEX `IMO_UNIQUE` (`IMO` ASC),
  CONSTRAINT `fk_ownShips_Ships1`
    FOREIGN KEY (`IMO`)
    REFERENCES `Mercantile Ships`.`Ships` (`IMO`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`rentalContracts`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`rentalContracts` (
  `registrationNo` INT NOT NULL AUTO_INCREMENT,
  `rentalCompanyIVA` VARCHAR(45) NOT NULL,
  `IMO` INT NOT NULL,
  `rentStartDate` DATETIME NULL,
  `rentEndDate` DATETIME NULL,
  `stipulationDate` DATETIME NULL,
  `availability` INT NULL,
  `eSignature` BLOB NULL,
  PRIMARY KEY (`registrationNo`),
  UNIQUE INDEX `registrationNo_UNIQUE` (`registrationNo` ASC),
  INDEX `fk_rentalContracts_rentalShips1_idx` (`IMO` ASC),
  INDEX `fk_rentalContracts_rentalCompanies1_idx` (`rentalCompanyIVA` ASC),
  CONSTRAINT `fk_rentalContracts_rentalShips1`
    FOREIGN KEY (`IMO`)
    REFERENCES `Mercantile Ships`.`rentalShips` (`IMO`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_rentalContracts_rentalCompanies1`
    FOREIGN KEY (`rentalCompanyIVA`)
    REFERENCES `Mercantile Ships`.`rentalCompanies` (`rentalCompanyIVA`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Customers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Customers` (
  `customerIVA` VARCHAR(45) NOT NULL,
  `CustomerName` VARCHAR(45) NULL,
  UNIQUE INDEX `customerIVA_UNIQUE` (`customerIVA` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`carriageContracts`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`carriageContracts` (
  `missionNo` INT NOT NULL,
  `customerIVA` VARCHAR(45) NOT NULL,
  `stipulationDate` DATETIME NULL,
  `price` INT NULL,
  `eSignature` BLOB NULL,
  `contractStatus` VARCHAR(45) NULL,
  INDEX `fk_carriageContract_Customer1_idx` (`customerIVA` ASC),
  UNIQUE INDEX `missionNo_UNIQUE` (`missionNo` ASC),
  CONSTRAINT `fk_carriageContract_Customer1`
    FOREIGN KEY (`customerIVA`)
    REFERENCES `Mercantile Ships`.`Customers` (`customerIVA`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Missions`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Missions` (
  `IMO` INT NOT NULL,
  `missionNo` INT NOT NULL,
  `missionStartDate` DATETIME NULL,
  `missionEndDate` DATETIME NULL,
  `missionLoad` INT NULL,
  `currCoordinates` VARCHAR(45) NULL,
  `advanceDelay` INT NULL,
  `missionValidity` BINARY(2) NULL DEFAULT 0,
  INDEX `fk_Missions_carriageContracts1_idx` (`missionNo` ASC),
  INDEX `fk_Missions_Ships1_idx` (`IMO` ASC),
  UNIQUE INDEX `missionNo_UNIQUE` (`missionNo` ASC),
  PRIMARY KEY (`missionNo`),
  CONSTRAINT `fk_Missions_carriageContracts1`
    FOREIGN KEY (`missionNo`)
    REFERENCES `Mercantile Ships`.`carriageContracts` (`missionNo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Missions_Ships1`
    FOREIGN KEY (`IMO`)
    REFERENCES `Mercantile Ships`.`Ships` (`IMO`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`haveCrew`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`haveCrew` (
  `crewType` INT NOT NULL,
  `IMO` INT NOT NULL,
  INDEX `fk_haveCrew_Ships1_idx` (`IMO` ASC),
  UNIQUE INDEX `IMO_UNIQUE` (`IMO` ASC),
  INDEX `crewTypeIndex` (`crewType` ASC),
  CONSTRAINT `fk_haveCrew_Ships1`
    FOREIGN KEY (`IMO`)
    REFERENCES `Mercantile Ships`.`Ships` (`IMO`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Crews`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Crews` (
  `pk` INT NOT NULL AUTO_INCREMENT,
  `crewType` INT NOT NULL,
  `role` VARCHAR(45) NULL,
  `qualifications` VARCHAR(45) NULL,
  `Certificate` BINARY(2) NULL,
  `vacancies` INT NULL,
  `salary` INT NULL,
  INDEX `fk_Crews_haveCrew1_idx` (`crewType` ASC),
  PRIMARY KEY (`pk`),
  UNIQUE INDEX `code_UNIQUE` (`pk` ASC),
  CONSTRAINT `fk_Crews_haveCrew1`
    FOREIGN KEY (`crewType`)
    REFERENCES `Mercantile Ships`.`haveCrew` (`crewType`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Routes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Routes` (
  `missionNo` INT NOT NULL,
  PRIMARY KEY (`missionNo`),
  CONSTRAINT `fk_Routes_Missions1`
    FOREIGN KEY (`missionNo`)
    REFERENCES `Mercantile Ships`.`Missions` (`missionNo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Points`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Points` (
  `missionNo` INT NOT NULL,
  `coordinates` VARCHAR(45) NULL,
  `expectedCrossing` DATETIME NULL,
  `effectiveCrossing` DATETIME NULL,
  CONSTRAINT `fk_Points_Routes1`
    FOREIGN KEY (`missionNo`)
    REFERENCES `Mercantile Ships`.`Routes` (`missionNo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Ports`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Ports` (
  `missionNo` INT NOT NULL,
  `portCoordinates` VARCHAR(45) NULL,
  `departureArrival` BINARY(2) NULL,
  `PortingDate` DATETIME NULL,
  `effectivelyPorted` BINARY(2) NOT NULL DEFAULT false,
  CONSTRAINT `fk_Ports_Routes1`
    FOREIGN KEY (`missionNo`)
    REFERENCES `Mercantile Ships`.`Routes` (`missionNo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Containers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Containers` (
  `missionNo` INT NOT NULL,
  `containerNo` INT NULL,
  `contentType` VARCHAR(45) NULL,
  `ownerData` VARCHAR(45) NULL,
  CONSTRAINT `fk_Loads_Missions1`
    FOREIGN KEY (`missionNo`)
    REFERENCES `Mercantile Ships`.`Missions` (`missionNo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`Employees`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`Employees` (
  `id` INT NOT NULL,
  `employeeName` VARCHAR(45) NULL,
  `qualifications` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`haveStaff`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`haveStaff` (
  `missionNo` INT NOT NULL,
  `id` INT NOT NULL,
  `role` VARCHAR(45) NULL,
  `qualifications` VARCHAR(45) NULL,
  INDEX `fk_haveStaff_Missions1_idx` (`missionNo` ASC),
  INDEX `fk_haveStaff_Employees1_idx` (`id` ASC),
  CONSTRAINT `fk_haveStaff_Missions1`
    FOREIGN KEY (`missionNo`)
    REFERENCES `Mercantile Ships`.`Missions` (`missionNo`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_haveStaff_Employees1`
    FOREIGN KEY (`id`)
    REFERENCES `Mercantile Ships`.`Employees` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Mercantile Ships`.`wherePorted`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Mercantile Ships`.`wherePorted` (
  `IMO` INT NOT NULL,
  `portCoordinates` VARCHAR(45) NOT NULL,
  `ported` BINARY(2) NOT NULL DEFAULT true,
  PRIMARY KEY (`IMO`),
  CONSTRAINT `fk_wherePorted_Ships1`
    FOREIGN KEY (`IMO`)
    REFERENCES `Mercantile Ships`.`Ships` (`IMO`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

USE `Mercantile Ships` ;

-- -----------------------------------------------------
-- procedure addMission
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addMission`(in n_missionNo int, in n_missionStartDate datetime, in n_missionEndDate
↪  datetime, in n_missionLoad int, in n_arrivalPortCoordinates varchar(45), in n_departurePortCoordinates varchar(45),
↪  in n_arrivalPortingDate datetime, in n_departureProtingDate datetime)
reads sql data
begin
        insert ignore into `Ships` values(-1, 'dummyShip', 0, 'martian', 'none', 'none');
        insert into `Mercantile Ships`.`Missions`(IMO, missionNo, missionStartDate, missionEndDate, missionLoad) values
        ↪  (-1, n_missionNo, n_missionStartDate, n_missionEndDate, n_missionLoad);
        call `addRoute`(n_missionNo);
    call `addPort`(n_missionNo, n_departurePortCoordinates, true, n_departureProtingDate);
    call `addPort`(n_missionNo, n_arrivalPortCoordinates, false, n_arrivalPortingDate);
    call `chooseProperShip`(n_missionNo, n_missionStartDate, n_missionEndDate, n_missionLoad, n_arrivalPortCoordinates,
    ↪  n_departurePortCoordinates, @c_IMO);
        call `pickStaff`(n_missionNo, @c_IMO, n_missionStartDate, n_missionEndDate);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addPort
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addPort` (in n_missionNo int, in portCoordinates varchar(45), in n_departureArrival
↪  binary(2), in n_portingDate datetime)
reads sql data
begin
        insert into `Mercantile Ships`.`Ports` values(n_missionNo, portCoordinates, n_departureArrival, n_portingDate,
        ↪  false);
end$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure addContainer
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addContainer` (in n_missionNo int, in n_containerNo int, in n_contentType
↪  varchar(45), in n_ownerData varchar(45))
reads sql data
begin
        insert into `Mercantile Ships`.`Containers` values (n_missionNo, n_containerNo, n_contentType, n_ownerData);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addPoint
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addPoint` (in n_missionNo int, in n_coordinates varchar(45), in n_expectedCrossing
↪  datetime)
reads sql data
begin
        insert into `Mercantile Ships`.`Points` values (n_missionNo, n_coordinates, n_expectedCrossing, null, null);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addRoute
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addRoute` (in n_missionNo int)
reads sql data
begin
        insert into `Mercantile Ships`.`Routes` values(n_missionNo);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyPoint
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyPoint` (in s_missionNo int, in s_coordinates varchar(45), in n_coordinates
↪  varchar(45), in n_expectedCrossing datetime)
reads sql data
begin
        update `Points` set coordinates = n_coordinates, expectedCrossing = n_expectedCrossing where missionNo =
        ↪  s_missionNo and coordinates like s_coordinates;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyPort
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyPort` (in s_missionNo int, in n_portCoordinates varchar(45), in
↪  n_departureArrival binary(2), in n_effectivelyPorted binary(2))
reads sql data
```

```
begin
        update `Ports` set portCoordinates = n_portCoordinates, departureArrival = n_departureArrival, effectivelyPorted
        ↪  = n_effectivelyPorted where missionNo = s_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteRoute
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteRoute` (in s_missionNo int)
reads sql data
begin
        delete from `Routes` where missionNo = s_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deletePort
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deletePort` (in s_missionNo int, in s_portCoordinates varchar(45))
reads sql data
begin
        delete from `Ports` where missionNo = s_missionNo and portCoordinates like s_portCoordinates;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deletePoint
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deletePoint` (in s_missionNo int, in s_coordinates varchar(45))
reads sql data
begin
        delete from `Points` where missionNo = s_missionNo and coordinates like s_coordinates;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure triggerCrossing
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`triggerCrossing` (in s_missionNo int, in s_coordinates varchar(45), in
↪  n_effectiveCrossing datetime)
reads sql data
begin
        update `Points` set effectiveCrossing = n_effectiveCrossing where missionNo = s_missionNo and coordinates like
        ↪  s_coordinates;
    set @expectedCrossing = (select expectedCrossing from `Points` where missionNo = s_missionNo and coordinates =
    ↪  s_coordinates);
        update `Missions` set advanceDelay = TIMESTAMPDIFF(MINUTE, @expectedCrossing, n_effectiveCrossing) where
        ↪  missionNo = s_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addOwnShip
-- -----------------------------------------------------
```

```
DELIMITER $$
USE `Mercantile Ships`$$
CREATE PROCEDURE `Mercantile Ships`.`addOwnShip`(in n_IMO int, in n_shipName varchar(45), in n_tonnage int, in
↪  n_nationality varchar(45), in n_draft varchar(45), in n_autonomy varchar(45), in n_crewType int, in
↪  n_portCoordinates varchar(45))
READS SQL DATA
BEGIN
     INSERT INTO `Mercantile Ships`.`Ships` VALUES(n_IMO, n_shipName, n_tonnage, n_nationality, n_draft, n_autonomy);
     INSERT INTO `Mercantile Ships`.`ownShips` VALUES(n_IMO);
     INSERT INTO `Mercantile Ships`.`haveCrew` VALUES(n_crewType, n_IMO);
     insert into `wherePorted` values(n_IMO, n_portCoordinates, true);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyOwnShip
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
CREATE PROCEDURE `Mercantile Ships`.`modifyOwnShip`(IN s_IMO INT, IN n_shipName VARCHAR(45), in n_tonnage int, in
↪  n_nationality varchar(45), in n_draft varchar(45), in n_autonomy varchar(45), in n_crewType int, in
↪  n_portCoordinates varchar(45), in n_ported binary)
READS SQL DATA
BEGIN
        UPDATE `Mercantile Ships`.`Ships` SET shipName = n_shipName, tonnage = n_tonnage, nationality = n_nationality,
        ↪  draft = n_draft, autonomy = n_autonomy WHERE IMO  = s_IMO;
        UPDATE `Mercantile Ships`.`haveCrew` set crewType = n_crewType where IMO = s_IMO;
    update `wherePorted` set portCoordinates = n_portCoordinates, ported = n_ported where IMO = s_IMO;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addRentalShip
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
CREATE PROCEDURE `Mercantile Ships`.`addRentalShip`(in n_IMO int, in n_shipName varchar(45), in n_tonnage int, in
↪  n_nationality varchar(45), in n_draft varchar(45), in n_autonomy varchar(45), in n_masterData varchar(45), in
↪  n_rentalCompanyContacts varchar(45), in n_crewType int, in n_portCoordinates varchar(45))
READS SQL DATA
BEGIN
     INSERT INTO `Mercantile Ships`.`Ships` VALUES(n_IMO, n_shipName, n_tonnage, n_nationality, n_draft, n_autonomy);
     INSERT INTO `Mercantile Ships`.`rentalShips` VALUES(n_IMO, n_masterData, n_rentalCompanyContacts);
     INSERT INTO `Mercantile Ships`.`haveCrew` VALUES(n_crewType, n_IMO);
     insert into `wherePorted` values(n_IMO, n_portCoordinates, true);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyRentalShip
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
CREATE PROCEDURE `Mercantile Ships`.`modifyRentalShip`(IN s_IMO INT, IN n_shipName VARCHAR(45), in n_tonnage int, in
↪  n_nationality varchar(45), in n_draft varchar(45), in n_autonomy varchar(45), in n_masterData varchar(45), in
↪  n_rentalCompanyContacts varchar(45), in n_crewType int, in n_portCoordinates varchar(45), in n_ported binary)
READS SQL DATA
BEGIN
        UPDATE `Mercantile Ships`.`Ships` SET shipName = n_shipName, tonnage = n_tonnage, nationality = n_nationality,
        ↪  draft = n_draft, autonomy = n_autonomy WHERE IMO  = s_IMO;
        UPDATE `Mercantile Ships`.`rentalShips` set masterData = n_masterData, rentalCompanyContacts =
        ↪  n_rentalComanyContacts where IMO = s_IMO;
    UPDATE `Mercantile Ships`.`haveCrew` set crewType = n_crewType where IMO = s_IMO;
    update `wherePorted` set portCoordinates = n_portCoordinates, ported = n_ported where IMO = s_IMO;
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure chooseProperShip
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`chooseProperShip`(in n_missionNo int, in n_missionStartDate datetime, in
↪  n_missionEndDate datetime, in s_missionLoad int, in n_arrivalPortCoordinates varchar(45), in
↪  n_departurePortCoordinates varchar(45), out c_IMO int)
reads sql data
begin
        declare s_IMO int;
    declare s_tonnage int;
    declare notRental int;
    declare shipNotAvailable int;
    declare freeShip int;
    declare missionConflict int;
    declare lastPortChecks int;
        DECLARE done INT DEFAULT FALSE;
        DECLARE allShips cursor for SELECT IMO, tonnage from `Ships` order by tonnage asc;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    open allShips;
    go: loop
                fetch allShips into s_IMO, s_tonnage;
                if done then
                        leave go;
                end if;
                if s_missionLoad <= s_tonnage then
                        select count(*) into notRental from(select * from `ownShips` where IMO = s_IMO) as nr;
            if notRental = 0 then
                                select count(*) into shipNotAvailable from (select rentStartDate, rentEndDate from
                                ↪  `rentalContracts` where IMO = s_IMO and ((rentStartDate > n_missionStartDate and
                                ↪  rentStartDate < n_missionEndDate) or (rentEndDate > n_missionStartDate and
                                ↪  rentEndDate < n_missionEndDate))) as t1;
            else
                                select 0 into shipNotAvailable;
                        end if;


                        select count(*) into freeShip from (select portCoordinates from `wherePorted` where
                        ↪  portCoordinates like n_departurePortCoordinates and IMO = s_IMO and IMO not in (select IMO
                        ↪  from `Missions` where ((missionEndDate > n_missionStartDate and missionEndDate <
                        ↪  n_missionEndDate) or (missionStartDate > n_missionStartDate and missionStartDate <
                        ↪  n_missionEndDate)) and missionNo != n_missionNo)) as fs;
                        if shipNotAvailable = 0 and freeShip > 0 then
                                update `Missions` set IMO = s_IMO where missionNo = n_missionNo;
                select s_IMO into c_IMO;
                leave go;
                        end if;

                        select count(*) into lastPortChecks from (select portCoordinates from `Ports` where portingDate
                        ↪  = (select max(portingDate) from `Ports` where DepartureArrival = false and missionNo in
                        ↪  (select missionNo from `Missions` where IMO = s_IMO) and portCoordinates like
                        ↪  n_departurePortCoordinates ) and missionNo !=n_missionNo) as t2;
                        select count(*) into missionConflict from (select * from `Ports` where (DepartureArrival = false
                        ↪  and portingDate > n_missionStartDate and portingDate < n_missionEndDate) and
                        ↪  (DepartureArrival = true and portingDate > n_missionStartDate and portingDate <
                        ↪  n_missionEndDate) and missionNo !=n_missionNo) as t3;
            if shipNotAvailable = 0 and missionConflict = 0 and lastPortChecks>0 then
                                update `Missions` set IMO = s_IMO where missionNo = n_missionNo;
                select s_IMO into c_IMO;
                leave go;
                        end if;

        end if;
        end loop go;
        close allShips;
end$$

DELIMITER ;
```

```
-- -----------------------------------------------------
-- procedure deleteOwnShip
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteOwnShip` (in s_IMO int)
reads sql data
begin
        delete from `ownShips` where IMO = s_IMO;
        delete from `Ships` where IMO = s_IMO;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteRentalShip
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteRentalShip` (in s_IMO int)
reads sql data
begin
        delete from `rentalShips` where IMO = s_IMO;
        delete from `Ships` where IMO = s_IMO;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addRentalContract
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addRentalContract` (in n_registrationNo int, in n_rentalCompanyIVA varchar(45), in
↪  n_IMO int, in n_rentStartDate datetime, in n_rentEndDate datetime, in n_eSignature blob)
reads sql data
begin
        insert into `rentalContracts` values(n_registrationNo, n_rentalCompanyIVA, n_IMO, n_rentStartDate,
        ↪  n_rentEndDate, null, null, n_eSignature);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyRentalContract
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyRentalContract` (in s_registrationNo int, in n_rentalCompanyIVA int, in n_IMO
↪  int, in n_rentStartDate datetime, in n_rentEndDate datetime, in n_eSignature blob)
reads sql data
begin
        update `rentalContracts` set registrationNo = s_registrationNo, rentalCompanyIVA = n_rentalCompanyIVA, IMO =
        ↪  n_IMO, rentStartDate = n_rentStartDate, rentEndDate = n_rentEndDate, eSignature = n_eSignature;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteRentalContract
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteRentalContract` (in s_registrationNo int)
reads sql data
begin
```

```
        delete from `rentalContracts` where registrationNo = s_registrationNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyMission
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyMission` (in s_missionNo int, in n_missionStartDate datetime, in
↪  n_missionEndDate datetime, in n_missionLoad int)
reads sql data
begin
        update `Mercantile Ships`.`Missions` set  missionStartDate = n_missionStartDate, missionEndDate =
        ↪  n_missionEndDate, missionLoad = n_missionLoad where missionNo = s_missionNo;
        call `addRoute`(s_missionNo);
    call `addPort`(s_missionNo, n_departurePortCoordinates, true, n_departureProtingDate);
    call `addPort`(s_missionNo, n_arrivalPortCoordinates, false, n_arrivalProtingDate);
    call `chooseProperShip`(s_missionNo, n_missionStartDate, n_missionEndDate, n_missionLoad, n_arrivalPortCoordinates,
    ↪  n_departurePortCoordinates, @c_IMO);
        call `pickStaff`(s_missionNo, @c_IMO, n_missionStartDate, n_missionEndDate);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure pickStaff
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`pickStaff` (in n_missionNo int, in n_IMO int, in n_missionStartDate datetime, in
↪  n_missionEndDate datetime)
reads sql data
begin
        DECLARE done INT DEFAULT FALSE;
        Declare s_id INT;
    declare s_qualifications varchar(45);
    declare s_role varchar(45);
        DECLARE idRange cursor for SELECT e.id, e.qualifications, c.role FROM `Employees` e inner join `Crews` c on
        ↪  c.crewType in (select crewType from `haveCrew` where IMO = n_IMO) and e.qualifications like
        ↪  c.qualifications;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN idRange;
    outer_loop: LOOP
                FETCH idRange into s_id, s_qualifications, s_role;
                IF done THEN
                        LEAVE outer_loop;
                END IF;
        employeeAvailability: begin
                        declare done2 INT DEFAULT FALSE;
            declare timeDescrepency int default false;
            declare filledVacancies int;
            declare totalVacancies int;
            declare MED datetime;
                        declare MEDs cursor for SELECT missionEndDate from `Missions` where missionNo in (select
                        ↪  missionNo from `haveStaff` where id = s_id);
            DECLARE CONTINUE HANDLER FOR NOT FOUND SET done2 = TRUE;
                        open MEDs;
                        inner_loop: loop
                                fetch MEDS into MED;
                                if done2 then
                                        leave inner_loop;
                                end if;
                                IF (MED > n_missionStartDate and MED < n_missionEndDate)  THEN
                                        set timeDescrepency = true;
                    leave inner_loop;
                                end if;
        end loop;
        if not timeDescrepency then
```

```
                        select vacancies into filledVacancies from `Crews` where qualifications like s_qualifications
                        ↪  and crewType = (select crewType from `haveCrew` where IMO = n_IMO);
                        select count(*) into totalVacancies from (select id from `haveStaff` where missionNo =
                        ↪  n_MissionNo and id = any (select id from `Employees` where qualifications=s_qualifications))
                        ↪  as idd;
            set @vacancyCheck := filledVacancies - totalVacancies;
            if @vacancyCheck > 0 then
                                INSERT INTO haveStaff (id, missionNo, role, qualifications) VALUES (s_id, n_missionNo,
                                ↪  s_role, s_qualifications);
                        end if;
                end if;
        end employeeAvailability;
        END LOOP;
        CLOSE idRange;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteMission
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteMission` (in s_missionNo int)
reads sql data
begin
    delete from `Ports` where missionNo = s_missionNo;
    delete from `Points` where missionNo = s_missionNo;
    delete from `Routes` where missionNo = s_missionNo;
    delete from `Containers` where missionNo = s_missionNo;
    delete from `haveStaff` where missionNo = s_missionNo;
        delete from `Missions` where missionNo = s_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyContainer
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyContainer` (in s_missionNo int, in n_containerNo int,  in n_contentType
↪  varchar(45), in n_ownerData varchar(45))
reads sql data
begin
        update `Mercantile Ships`.`Containers` set containerNo = n_containerNo, contentType = n_contentType, ownerData =
        ↪  n_ownerData where missionNo = n_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteContainer
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteContainer` (in s_missionNo int, in s_containerNo int)
reads sql data
begin
        delete from `Containers` where missionNo = s_missionNo and containerNo = s_containerNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addCarriageContract
-- -----------------------------------------------------

DELIMITER $$
```

```
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addCarriageContract` (in n_missionNo int, in n_customerIVA varchar(45), in n_price
↪  int, in n_eSignature blob, in n_contractStatus varchar(45))
reads sql data
begin
        insert into `carriageContracts` values(n_missionNo, n_customerIVA, now(), n_Price, n_eSignature,
        ↪  n_contractStatus);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyCarriageContract
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyCarriageContract` (in s_missionNo int, in n_customerIVA varchar(45), in
↪  n_price int, in n_eSignature blob, in n_contractStatus varchar(45))
reads sql data
begin
        update `carriageContracts` set customerIVA = n_customerIVA, stipulationDate = now(), price = n_Price, eSignature
        ↪  = n_eSignature, contractStatus = n_contractStatus where missionNo = s_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addCustomer
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addCustomer`(in n_customerIVA varchar(45), in n_customerName varchar(45))
reads sql data
begin
        insert into `Customers` values(n_customerIVA, n_customerName);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyCustomer
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyCustomer`(in s_customerIVA varchar(45), in n_customerName varchar(45))
reads sql data
begin
        update `Customers` set  customerName = n_customerName where customerIVA like s_customerIVA;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteCustomer
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteCustomer`(in s_customerIVA varchar(45))
reads sql data
begin
        declare customerMission int;
        declare done int default false;
        declare customerMissions cursor for select missionNo from  `carriageContracts` where customerIVA like
        ↪  s_customerIVA;
        declare continue handler for not found set done = true;
        open customerMissions;
        go: loop
                fetch customerMissions into customerMission;
                if done then
                        leave go;
                end if;
                call `deleteCarriageContract`(customerMission);
                call `deleteMission`(customerMission);
        end loop go;
        close customerMissions;
        delete from `Customers` where customerIVa like s_customerIVA;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addRentalCompany
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addRentalCompany` (in n_rentalCompanyIVA varchar(45), in n_rentalCompanyName
↪  varchar(45))
reads sql data
begin
        insert into `rentalCompanies` values (n_rentalCompanyIVA, n_rentalCompanyName);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyRentalCompany
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyRentalCompany` (in n_crewType int, in s_IMO int)
reads sql data
begin
        update `rentalCompanies` set crewType = n_crewType where IMO = s_IMO;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteRentalCompany
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteRentalCompany` (in s_rentalCompanyIVA varchar(45))
reads sql data
begin
        delete from `rentalCompanies` where  rentalCompanyIVA like s_rentalCompanyIVA;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addEmployee
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addEmployee` (in n_id int, in n_employeeName varchar(45), in n_qualifications
↪  varchar(45))
reads sql data
begin
        insert into `Employees` values(n_id, n_employeeName, n_qualifications);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyEmployee
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyEmployee` (in s_id int, in n_employeeName varchar(45), in n_qualifications
↪  varchar(45))
reads sql data
begin
        update `Employees` set employeeName = n_employeeName, qualifications = n_qualifications where id = s_id;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteEmployee
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteEmployee` (in s_id int)
reads sql data
begin
        delete from `Employees` where id = s_id;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure detectCollision
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`detectCollision` (in n_missionNo int, in n_missionStartDate datetime)
reads sql data
begin
        declare done1 int default false;
        declare s_missionNo int;
        declare s_missionStartDate int;
        declare s_missionEndDate int;
        declare e_Missions cursor for select missionNo, missionStartDate, missionEndDate from `Missions` where
        ↪  missionEndDate > n_missionStartDate and TIMESTAMPDIFF(MINUTE,n_missionStartDate,missionEndDate) < 60;
    declare continue handler for not found set done1 = true;
    open e_Missions;
    go: loop
                fetch e_Missions into s_missionNo, s_missionStartDate, s_missionEndDate;
                if done1 then
                        leave go;
                end if;
        set @CollisionNo = (select count(*) from(select * from `Ports` inner join (select * from `Ports` where missionNo
        ↪  = n_missionNo) as this where missionNo = s_missionNo and portCoordinates like this.portCoordinates ) as
        ↪  collisions);
        if @CollisionNo > 0 then
                        set @message = concat(@CollisionNo, ' risks of collision detected');
                        signal sqlstate '45000' set message_text = @message;
                end if;
        end loop;
    close e_Missions;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure calculateTurnover
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`calculateTurnover`(out turnover int)
reads sql data
begin
        declare wins int default 0;
        declare done int default false;
    declare s_price int;
    declare e_cc cursor for select price from `carriageContracts` where timestampdiff(day,stipulationDate, now()) < 30;
    declare continue handler for not found set done = true;
    open e_cc;
    go: loop
                fetch e_cc into s_price;
        if done then
                        leave go;
                end if;
        set wins = wins + s_price;
    end loop;
    close e_cc;
    set turnover = wins;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure checkPersonnelNo
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`checkPersonnelNo`(in s_missionNo int)
reads sql data
begin
        set @personnelCount = (select COUNT(*) from (select id from `haveStaff` where missionNo = s_missionNo) as idd);
    if @personnelCount > 30 or @personnelCount < 20 then
                signal sqlstate '45000' set message_text = 'irregular crew count';
        end if;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure setMissionValidity
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`setMissionValidity`(in s_missionNo int)
reads sql data
begin
        declare s_IMO int;
        set s_IMO = (select IMO from `Missions` where missionNo = s_missionNo);
        update `Missions` set missionValidity = true where IMO = s_IMO and missionLoad > 80*(select tonnage from `Ships`
        ↪  where IMO = s_IMO)/100;
        update `Missions` set missionValidity = false where IMO = s_IMO and missionLoad < 80*(select tonnage from
        ↪  `Ships` where IMO = s_IMO)/100;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure triggerEffectivePorting
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`triggerEffectivePorting`(in n_missionNo int, in s_IMO int, in n_portCoordinates
↪  varchar(45))
reads sql data
begin
        update `Ports` set portingDate = now(), effectivelyPorted = true where missionNo = n_missionNo and
        ↪  portCoordinates like n_portCoordinates;
        set @departureArrival = (select count(*) from (select departureArrival from `Ports` where missionNo =
        ↪  n_missionNo and portCoordinates like n_portCoordinates and departureArrival = false) as da);
        if @departureArrival = 1 then
        update `wherePorted` set portCoordinates = n_portCoordinates, ported = true where IMO = s_IMO;
        end if;
end$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure checkMissionCount
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`checkMissionCount`()
reads sql data
begin
        set @missionCount = (select count(*) from (select missionStartDate from `Missions` where (TIMESTAMPDIFF(month,
        ↪  missionStartDate,  now())<=12)) as mC);
    if @missionCount > 110 then
                signal sqlstate '45000' set message_text = 'you are exceeding the missions quota of the year';
        end if;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addCrewTypeAssociation
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addCrewTypeAssociation` (in n_crewType int, in n_IMO int)
reads sql data
begin
        insert into `haveCrew` values (n_crewType, n_IMO);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyCrewTypeAssociation
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyCrewTypeAssociation` (in n_crewType int, in s_IMO int)
reads sql data
begin
        update `haveCrew` set crewType = n_crewType where IMO = s_IMO;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteCrewTypeAssociation
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteCrewTypeAssociation` (in s_IMO int)
reads sql data
begin
        delete from `haveCrew` where  IMO = s_IMO;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure addCrewTypeRole
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`addCrewTypeRole` (in n_crewType int, in n_role varchar(45), in n_qualifications
↪  varchar(45), in n_certificate binary(2), in n_vacancies int, in n_salary int)
reads sql data
```

```
begin
	insert into `Crews` values(n_crewType, n_role, n_qualifications, n_certificate, n_vacancies, n_salary);
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteCarriageContract
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteCarriageContract`(in s_missionNo int)
reads sql data
begin
	call `deleteMission`(s_missionNo);
	delete from `Mercantile Ships`.`carriageContracts` where missionNo = s_missionNo;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure modifyCrewTypeRole
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`modifyCrewTypeRole` (in s_crewType int, in n_role varchar(45), in n_qualifications
↪  varchar(45), in n_certificate binary(2), in n_vacancies int, in n_salary int)
reads sql data
begin
	update`Crews` set role = n_role, qualifications = n_qualifications, certificate = n_certificate, vacancies =
	↪  n_vacancies, salary = n_salary where crewType = n_crewType;
end$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure deleteCrewTypeRole
-- -----------------------------------------------------

DELIMITER $$
USE `Mercantile Ships`$$
create procedure `Mercantile Ships`.`deleteCrewTypeRole` (in s_crewType int)
reads sql data
begin
	delete from `Crews` where crewType = s_crewType;
end$$

DELIMITER ;
USE `Mercantile Ships`;

DELIMITER $$
USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`checkRentalShipsNumber` BEFORE INSERT ON `rentalShips` FOR
↪  EACH ROW
BEGIN
	if (select count(*) from (select * from `rentalShips` where IMO in (select IMO from `Missions` where
	↪  missionEndDate > NOW())) as shipsNumber) > 20 then
		signal sqlstate '45000'
	SET MESSAGE_TEXT = 'you can not have more than 20 active rental ships';
	end if;
	if (select count(*) from (select * from `ownShips` where IMO = new.IMO) as shipsNumber) > 0 then
		signal sqlstate '45000'
	SET MESSAGE_TEXT = 'this is a company ship';
	end if;
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`checkOwnShipsNumber` BEFORE INSERT ON `ownShips` FOR EACH ROW
BEGIN
	if (select count(*) from (select * from `ownShips`) as shipsNumber) > 10 then
```

```
                signal sqlstate '45000'
        SET MESSAGE_TEXT = 'you can not insert more than 10 ships';
        end if;
    if (select count(*) from (select * from `rentalShips` where IMO = new.IMO) as shipsNumber) > 0 then
                signal sqlstate '45000'
        SET MESSAGE_TEXT = 'this is a rental ship';
        end if;
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`addRenatlShipAvailability` BEFORE INSERT ON `rentalContracts`
↪  FOR EACH ROW
BEGIN
        set new.availability = TIMESTAMPDIFF(hour, new.rentStartDate, new.rentEndDate);
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`addStipulationDate` BEFORE INSERT ON `rentalContracts` FOR
↪  EACH ROW
BEGIN
        set new.stipulationDate = NOW();
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`updateStipulationDate` BEFORE UPDATE ON `rentalContracts` FOR
↪  EACH ROW
BEGIN
        set new.stipulationDate = NOW();
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`updateRenatlShipAvailability` BEFORE UPDATE ON
↪  `rentalContracts` FOR EACH ROW
BEGIN
        set new.availability = TIMESTAMPDIFF(hour, new.rentStartDate, new.rentEndDate);
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`checkPortsNum` BEFORE INSERT ON `Ports` FOR EACH ROW
BEGIN
        if ((select count(*) from (select * from `Ports` where missionNo = new.missionNo) as fc) > 2) then
                set new.missionNo = null;
        end if;
END$$

USE `Mercantile Ships`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Mercantile Ships`.`calculateMissionLoad` AFTER INSERT ON `Containers` FOR EACH
↪  ROW
BEGIN
        update `Missions` set missionLoad = missionLoad + 1;
END$$


DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Bibliography

[1] **MySQL 5.7 Reference Manual**: https://downloads.mysql.com/docs/refman-5.7-en.a4.pdf

[2] **MySQL Workbench 6.3 Reference Manual:** https://downloads.mysql.com/docs/workbench-en.a4.pdf