

Exercice : Portes Conteneurs

# Contexte



Le transport maritime mondial passe aujourd’hui par les [conteneurs](https://fr.wikipedia.org/wiki/Conteneur), grosses boite d’acier de 6 à 12m de long pour le modèle standard empilé sur des [navires](https://fr.wikipedia.org/wiki/Porte-conteneurs) qui desservent tout les ports du monde. La plus longue ligne maritime relie Boston à San-Francisco en passant par la Méditerranée, le canal de Suez, Singapour, Taïwan et le Japon en moins de 15 semaines.

L’organisation du transport maritime se résume à associer des conteneurs aux navires qui passent par les ports de destination et à empiler les conteneurs dans le bon ordre. Rien de tel qu’une base de données relationnelle pour gérer tout ça !

# Enoncé

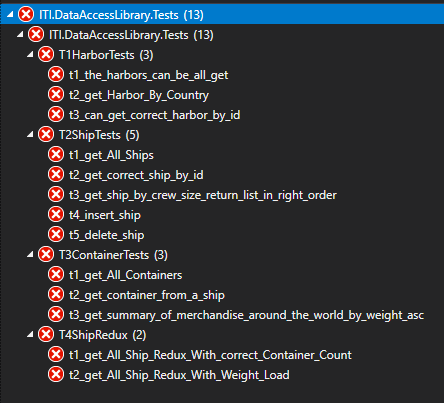
L'objectif de cet exercice est d'implémenter une librairie d’accès à une base de données relationnelle (DAL, Data Access Library *ou* Layer) sous la forme de tests unitaires. Autrement dit : l'objectif de l'exercice est de faire passer en vert un jeu de tests.

Vous disposez pour cela d'une solution comprenant 2 projets :

* ITI.DataAccesLibrary.Tests, contient les tests unitaires
* ITI. DataAccesLibrary, une implémentation minimale du modèle (le code écrit ne permet rien de plus que de pouvoir compiler sans erreur)

Pour faire tourner les tests unitaires il vous suffit d’utiliser le *Test Explorer*.

Comme vous pouvez le constater, pour le moment, tous les tests sont rouges.



Pour les autres tests, à vous de faire en sorte qu'ils passent en vert. Pour cela, vous avez le droit de faire ce que bon vous semble dans le projet ITI.DataAccesLibrary (modifier l'implémentation des méthodes, des propriétés ou des constructeurs, ajouter des membres *internal* ou *private*, créer de nouveaux types etc...) En revanche, vous ne devez pas modifier le projet ITI.DataAccesLibrary.Tests.

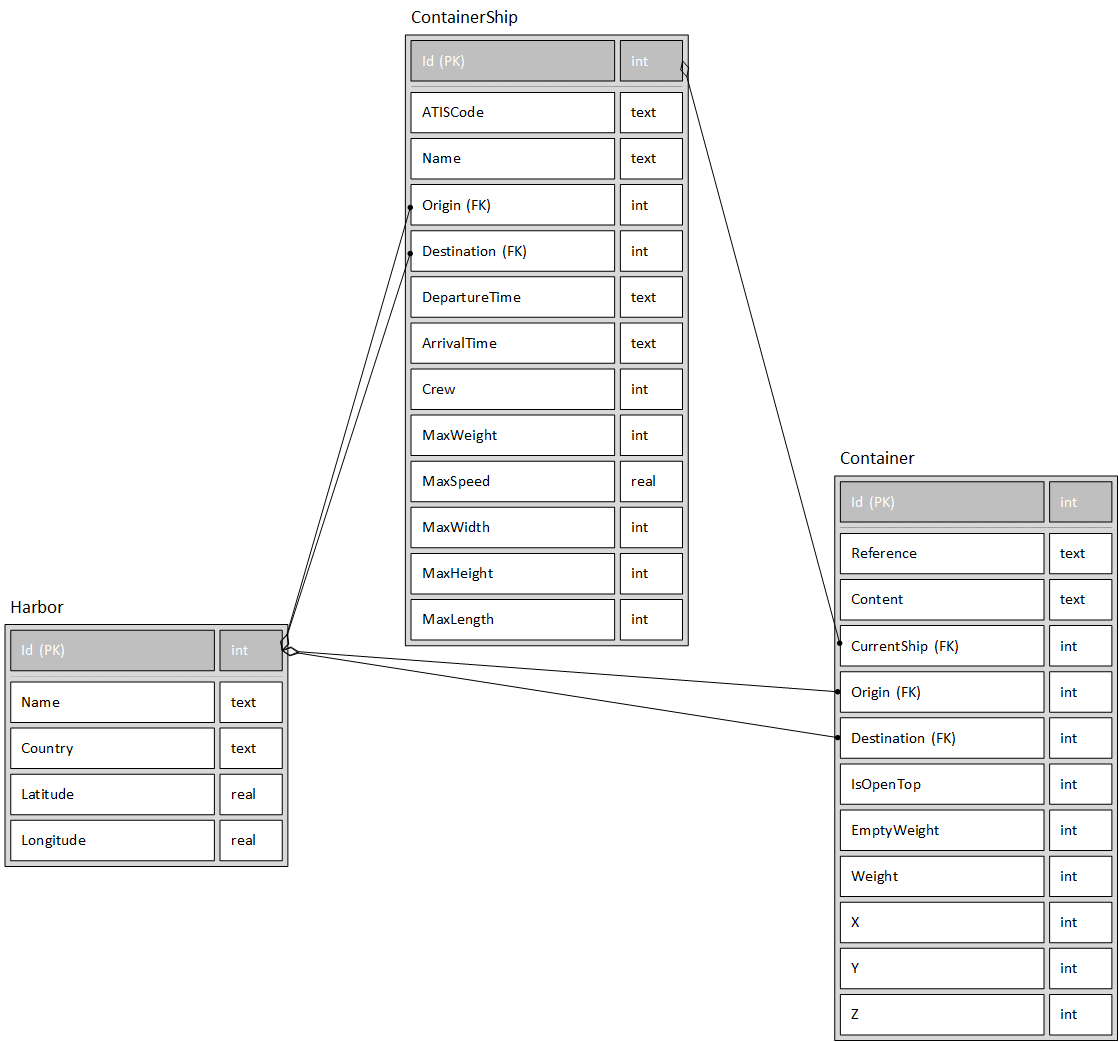
La librairie est composée de classes qui reflètent les éléments qu’elle récupère. Par exemple HarborQueries gère les éléments de type Harbor, etc. Chacune de ces classes dérive de la classe abstraite Queries qui expose la propriété ConnexionString qui décrit le chemin vers la base SQLite généré par les tests. C’est elle qui faudra appeler pour instancier les connexions à SQLite.

Toutes les requêtes sont décrites dans les classes de ITI.DataAccessLibrary, les commentaires additionnels précisent au besoin les paramètres et la sortie. Vous pouvez choisir de charger en mémoire chaque table et d’effectuer toutes les opérations dans la DAL (pas sûr que ce soit le plus efficace niveau mémoire…) ou d’écrire les requêtes spécifiques à chaque fois. Toutes les requêtes sont réalisables directement en SQL et ne nécessitent normalement pas de manipulation (Sort, LINQ supplémentaires).

A noter que pour forger vos requêtes, vous pouvez utiliser l’interpolation (utiliser directement les données d’entré) mais s’est s’exposer à des injections SQL. Penchez-vous sur les requêtes préparés présente dans le package System.Data.SQLite utilisé.

## Structure de la base

Pour faciliter l’embarquement d’une base de données, le choix s’est porté sur SQLite, une version allégée et portable de SQL. La base modélise 3 table liée entre elles. A chaque exécution des tests elle est regénéré de manière aléatoire et cohérente. Inutile donc de se baser sur son contenu. (Qui reste néanmoins accessible via une invite de commande SQLite)



Les ports (Harbor) sont modélisés simplement avec un id, un nom, un pays et une position (Latitude/Longitude)

Les navires (ContainerShip) sont modélisés par un id, un code [ATIS](http://www.permisbateauonline.com/permis-fluvial/tutoriaux/code-ATIS-des-equipements-VHF.php) (une Guid par soucis de simplicité), le nom, le port d’origine et de destination (relation un à plusieurs depuis les ports), la date de départ et d’arrivée (Au format supporté par SQLlite), du nombre de membre d’équipage, de la charge maximale (en Tonnes) et la vitesse maximale du navire (en nœuds nautiques). Enfin 3 valeurs définissent le nombre de conteneur à bord. Par convention la largeur (width) comme axe X, la hauteur (height) comme axe Y et la longueur (length) comme axe Z.



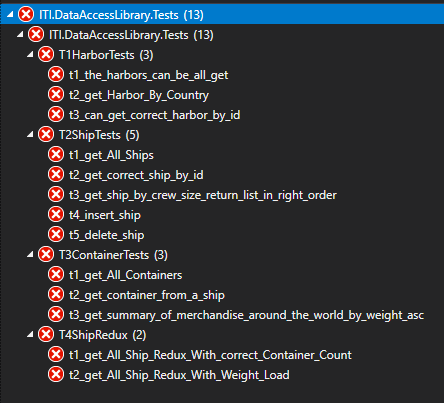
Les conteneurs (Container) sont modélisés par un id, un code de référence (Guid), un contenu, le navire qui les transporte (relation un à plusieurs depuis les navires), l’origine et la destination (relation un à plusieurs depuis les ports), si le conteneur à un toit ouvert (il est donc forcément au sommet des piles), sa masse à vide, sa charge et enfin sa position dans le navire (cf : convention plus haut).

## Fonctionnalités attendues

Les tests unitaires sont là pour spécifier de façon détaillée les fonctionnalités attendues. Cependant voici une courte description de ce qui est attendu :

* Port
  + On peut récupérer tous les ports.
  + On peut récupérer un port spécifique.
  + On peut récupérer tous les ports d’un pays
* Bateaux
  + On peut créer un bateau
  + On peut supprimer un bateau
  + On peut récupérer tous les bateaux
  + On peut récupérer un bateau spécifique.
  + On peut récupérer tous les bateaux triés par équipage
  + On peut récupérer tous les bateaux triés par date de départ
  + On peut récupérer tous les bateaux triés par capacité
  + On peut récupérer tous les bateaux qui sont vides
  + On peut récupérer tous les bateaux qui transportent un contenu spécifique
  + On peut récupérer une synthèse des navires (total de conteneur transporté et masse total de marchandise à bord)
* Conteneurs
  + On peut récupérer tous les conteneurs
  + On peut récupérer la liste des conteneurs d'un navire spécifique
  + On peut récupérer une synthèse des marchandises mondiales, du poids total transporté et du nombre de conteneur (par ordre de poids ascendant)

Il y a 13 tests à passer au vert :



# Ressources

<https://www.sqlite.org>

<https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki>

Au boulot !