# TP2. Implémentation du model SQL.

## Objectifs :

* Installation de SQL Serveur express et de l’outil SSMS.
* Configuration de la base de données MyAirport.
* Implémentation d’un second modèle de données.
* Utilisation Sql depuis le code C#.

## Déroulement :

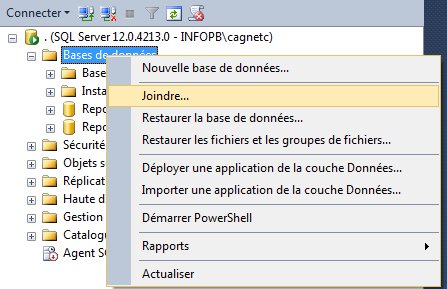
### Installation d’un serveur SQL.

Commencer par l’installation de SQL Express with Advanced Services.

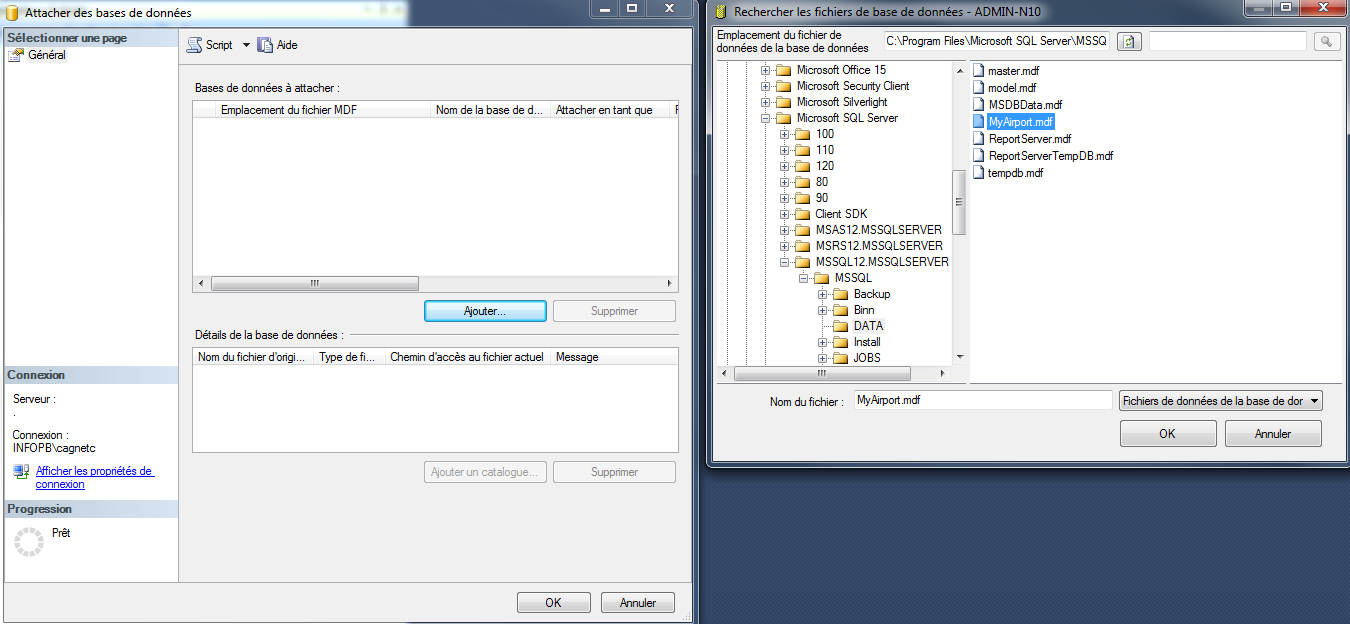
Récupérer sur le site du cours la Base de données MyAirport

Décompresser et copier les fichiers de bases de données (Mdf et ldf) dans le répertoire c:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\ le nom du répertoire peut varier en fonction de la version de Sql Server installée.

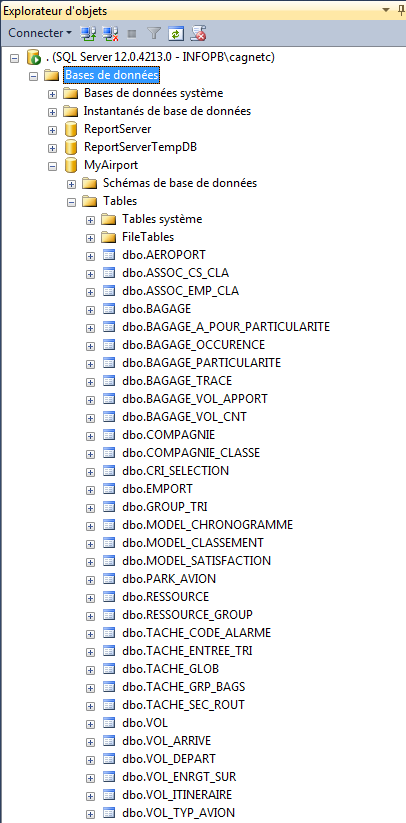
Démarrer SSMS (Sql Server Management Studio) et faites un clic droit sur le répertoire ‘Bases de données’ et choisir Joindre…



Cliquez sur le bouton Ajouter et rechercher le fichier MyAirport.mdf.



Enfin vérifier que la base de données a été correctement chargée, en déroulant le répertoire Tables de la base MyAirport, vous devriez voir des tables BAGAGE, MODEL, TACHE et VOL.



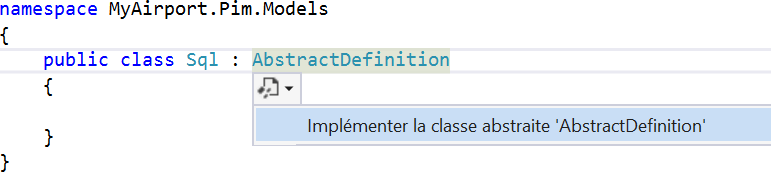
### Création d’un nouveau modèle de données.

Cette section décrit comment ajouter un nouveau modèle de données à l’application MyAirport, ce nouveau modèle basé sur SQL remplacera le modèle Natif initialement créé. Le choix de l’un ou l’autre des modèles se fera toujours via le fichier de configuration de l’application.

Ajouter un nouveau projet Models.Sql de type bibliothèque de classe dans le répertoire Models.

Et ajouter les références aux projets Models.AbstractDefinition et Entities.

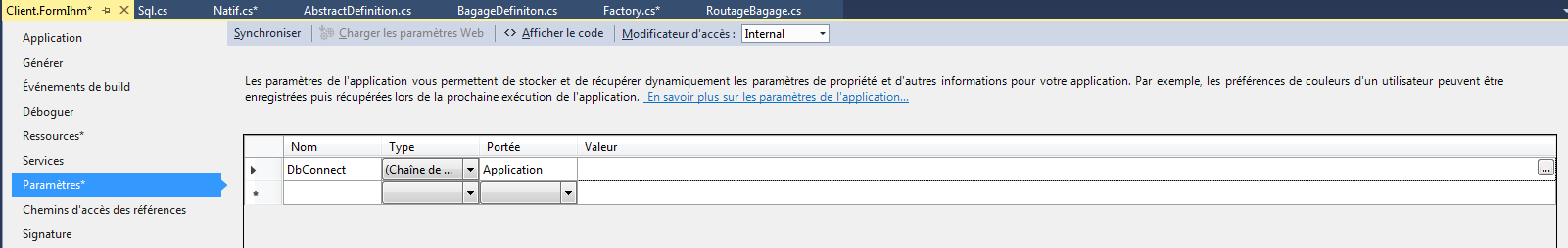
Créer une classe Sql qui dérive de la classe AbstractDefinition et implémenter l’ensemble des méthodes abstraites via l’utilisation des smart tags :



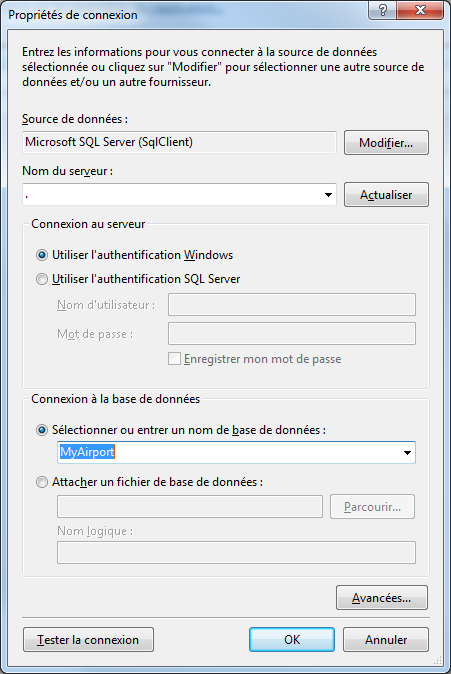
Les méthodes abstraites de la classe AbstractDefinition sont automatiquement générées avec un code levant une exception par défaut.

La classe SQL nécessite l’utilisation d’une chaine de connexion pour accéder à la base de données, ce paramétrage peut être inscrit en dur dans le code mais cela implique de devoir recompiler l’application à chaque changement d’adresse de la base de données. Il a déjà été utilisé dans le projet Model.Factory une méthode permettant l’accès au fichier de configuration app.config. Cette solution va de nouveau être utilisée afin de passer la chaine de connexion à notre application.

Dans le projet Client.FormIhm effectuer un clic droit sur le projet.et sélectionner ‘Propriétés’ et puis l’onglet ‘Paramètres’.



Ajoutez un nouveau paramètre ‘DbConnect’ de type chaine de connexion et dans le champ valeur cliquez sur le bouton de recherche ‘…’



Indiquez l’adresse du serveur SQL :

‘.’ Ou ‘localhost’ pour des installations SQL SERVER en local

‘.\SQLEXPRESS’ pour des versions SQL Express.

Puis sélectionner la base de données sur laquelle le client aura l’accès par défaut.

Visualiser le fichier app.config pour voir les changements effectués par Visual Studio.

<connectionStrings>

<add name="Client.FormIhm.Properties.Settings.DbConnect" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=MyAirport;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

Modifier la valeur de la propriété Name en MyAiport.Pim.Settings.DbConnect

Ajouter Model.Sql en référence à Model.Factory, puis recompiler votre projet.

### Implémentation SQL.

Dans SSMS écrire et tester la requête permettant de récupérer les colonnes nécessaires à la création d’un objet BagageDefinition.

Cette requête va exploiter les données des tables BAGAGE, BAGAGE\_A\_POUR\_PARTICLARITE et BAGAGE\_PARTICULARITE.

L’ensemble des classes nécessaires à la manipulation d’objet SQL est regroupé dans la librairie System.Data et principalement dans le namespace System.Data.SqlClient.

Pour jouer une requête sur un serveur SQL il est nécessaire de disposer d’une connexion à la base de données (via l’objet SqlConnection) et d’une commande (via l’objet SqlCommand). L’objet commande va s’appuyer sur l’objet connexion pour discuter avec le serveur SQL. Une connexion ne pouvant pas servir simultanément à l’exécution de 2 requêtes, une instance SqlConnection devra être créée pour chaque commande et être fermée le plus rapidement possible. Bien que ce mécanisme puisse paraitre lourd dans la gestion, il est dans la réalité optimisé par un mécanisme de pool de connexions regroupées par chaine de connexion. <https://msdn.microsoft.com/fr-fr/library/8xx3tyca(v=vs.110).aspx>

Dans la classe Sql ajouter 2 champs privés strcnx et commandGetBagage qui contiendront respectivement la chaine de connexion à la base de données et la requête à exécuter (A vous de construire cette requête, le select implémenté dans le TP vous permet de connaitre l’ensemble des colonnes devant-être sélectionnées).

string strCnx = ConfigurationManager.ConnectionStrings["MyAiport.Pim.Settings.DbConnect"].ConnectionString;

string commandGetBagageIata =

"SELECT b.ID\_BAGAGE, b.CODE\_IATA, b.COMPAGNIE, b.LIGNE, b.DATE\_CREATION, b.ESCALE, b.CLASSE, b.CONTINUATION, cast(iif(bp.PARTICULARITE is null, 0, 1) as bit) as 'RUSH' FROM BAGAGE b " + "…"

string commandGetBagageId = …

Ensuite implémenter les 2 méthodes GetBagage

Par exemple pour GetBagage(string codeIata) de la manière suivante :

public override BagageDefinition GetBagage(int idBagage)

{

BagageDefinition bagRes = null;

using (SqlConnection cnx = new SqlConnection(strCnx))

{

SqlCommand cmd = new SqlCommand(this.commandGetBagageId, cnx);

cmd.Parameters.AddWithValue("@id", idBagage);

cnx.Open();

//Implémenter ici le code de récupération et de convertion

}

return bagRes;

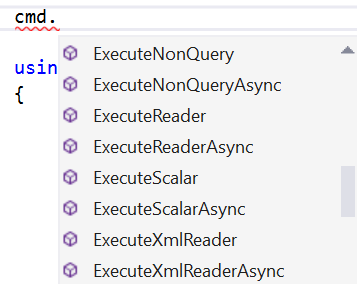
}

Remarquer ici l’utilisation de cmd.Parameters qui va servir à fournir l’identifiant en paramètre de la commande. L’utilisation des paramètres pour passer des valeurs à des instructions SQL permet la vérification et la validation des types. Contrairement au texte de la commande, l’entrée de paramètre est traitée comme une chaine littérale et non pas comme du code exécutable. Cela permet de se protéger des attaques par injection de code SQL. Les performances d’exécution sont aussi améliorées coté serveur SQL, car celui-ci associe la requête à un unique plan d’exécution mis en cache, quelque soit la valeur des paramètres. Attention ne jamais utiliser de concaténation de string pour construire les requêtes SQL du genre :



Lecture et mise en forme des résultats de la commande.

Utiliser les méthodes Execute…. De la classe SqlCommand pour pouvoir exécuter les requetés a vous de choisir la bonne méthode en fonction du résultat attendu par l’exécution de la requête.



ExecuteNonQuery exécute une instruction QL et retourne le nombre de lignes affectées par l’instruction, cette méthode est utilisée pour les requêtes Insert/Update/Delete par exemple.

ExecuteReader est utilisée pour la lecture de résultats sous forme de tableau, au moins 1 ligne de deux colonnes ou plusieurs lignes. Le résultat est stocké dans un SqlDataReader qui permet une lecture ligne par ligne des données retournées par l’instruction SQL.

ExecuteScalar exécute la requête et retourne la première colonne de la première ligne du jeu de résultat, cette méthode est particulièrement utile dans les instructions de count par exemple est présente l’avantage d’être beaucoup plus légère que l’utilisation d’un SqlDataReader.

Vous allez devoir traiter chacune des colonnes retournées par votre requête afin de construire un objet BagageDefinition.

Il est possible de traiter chaque champ en adaptant le code ci-dessous :

bagRes.CodeIata = sdr.GetString(sdr.GetOrdinal("code\_iata"));

Il vous sera juste nécessaire d’adapter le nom de la propriété, le type de donnée et le nom de la colonne.

Attention

Il est important de gérer les cas où les champs peuvent être null dans Sql (type DBnull) car cela générera une exception sinon. De plus certain champ devront être converti manuellement. Le champ PRIORITAIRE peut contenir n’importe quel caractère seul le ‘Y’ indique un bagage non prioritaire (Voir options n°2 en fin de TP), il convient donc de ‘transformer’ ce caractère en un type bool pour pouvoir créer l’objet BagageDefinition. Le traitement peut être fait dans le code c# ou dans la requête SQL.

Pour les cas ou plusieurs lignes sont retournées par la requête utiliser la méthode Read de SqlDataReader afin de lire ligne par ligne les résultats retournés.

Attention la méthode GetBagage(string CodeIata) renvoie une liste de bagageDefinition. Il convient donc ici de créer un nouveau bagageDefinition pour chaque ligne retournée et de renvoyer la liste des bagages à la méthode appelante.

### Exécution et validation du fonctionnement.

Ajouter un second bouton à la form Client et associer ce bouton à l’appel de la méthode GetBagage(int id)

Dans la classe Factory ajouter la prise en compte du modèle Sql.

Penser à modifier la valeur de la balise Factory à Sql pour tester avec le bon modèle de de données.

Placer des points d’arrêt sur chacune des méthodes associées aux boutons. Compiler et générer l’application et vérifier que les deux méthodes fonctionnement correctement et renvoient bien un résultat.

## Options de développement :

1. Généralement les utilisateurs ne saisissent que 6 des 12 digit du code iata par exemple pour le code '023232546100' les utilisateurs ne saisissent que '325461' les 4 premiers digit étant associé à un prestataire et les deux derniers étant toujours 0. Vous pouvez implémenter cette option facilement dans votre requête SQL.
2. La valeur du champ continuation est un unique caractère toutes les compagnies ont leur propre signification il existe donc une table [dbo].[COMPAGNIE\_CLASSE] qui sert à définir pour un caractère et une compagnie donnée s’il s’agit d’un bagage Prioritaire. La Table BAGAGE fait référence au code IATA de la compagnie il est donc nécessaire de mettre en relation la colonne CONPAGNIE de la table BAGAGE avec la colonne CODE\_IATA de la table compagnie, cela permet de récupérer l’identifiant de la compagnie pour trouver dans la table COMPAGNIE\_CLASSE si tel ou tel caractère est défini comme prioritaire.
3. L’option n°2 permet également d’afficher le nom véritable de la compagnie et non pas son code 2 lettres.

## Questions :

* Qu’elle est l’utilité de l’utilisation de l’instruction using lors de la création d’une instance SQLConnection ?
* Que se passe-t-il lors de l’appel de la méthode close sur l’objet SqlConnection ?
* Quelles sont les avantages à l’utilisation des paramètres dans les commandes SQL ?
* Pourquoi les valeurs null SQL sont retournées comme des objet de type DBNull et non pas null ?