Projet : DataCloner

# Vue d’ensemble

C’est un algorithme récursif qui se base sur une ligne de départ d’une base de données pour aller récupérer toutes les dépendances et dérivées de celle-ci. Au final, c’est un enregistrement intègre qui peut être enregistré dans un fichier pour un chargement futur ou directement dans une base de donnée comportant la même structure de tables.

# À qui s’adresse l’application?

* À toutes les entreprises possédant une base de données.
* Tous les développeurs, analystes et les gens du pilotage devant tester des cas en problème de production.
* Pour les dévelppeurs voulant automatiser des domaines d’essais pour ainsi gagner riduculement de temps.

# Options

* Support entre multi-serveurs source et destination.
* Duplications entre bases de données.
* Enregistrement dans un fichier.
* Chargement depuis un fichier.
* Liaisons configurables entre tables et bases de données.
* Chargement avec ou sans dérivées.
* Chargement d’une à plusieurs lignes depuis une à plusieurs tables de départ.
* Utilisable en ligne de commande.
* Support de tables sources/destinations similaires (colonnes en plus ou moins)
* Brouillage des données sensibles

# Analyse de cas

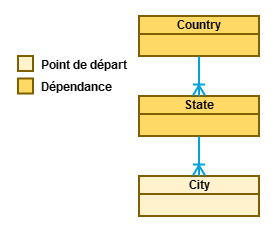
### Notions de base

Une clef primaire et étrangère identifient de façon unique un enregistrement d’une table. La clef peut être composée de plusieurs colonnes.

### Principe de base

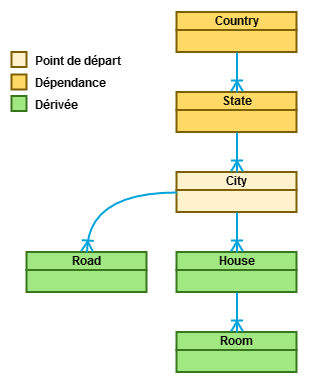
Chaque enregistrement effectué dans la base de données de destination est conservé dans un dictionnaire dont la clef est composé de “ServerId, Database, Schema, Table, PK\_source“ et la valeur est la nouvelle PK. De cette façon, on élimie les doublons en vérifiant d’abors si la ligne a déjà été enregistrée et si oui, on utilise la PK du dictionnaire sinon, on crer la ligne. La création implique la récupération des dépendances.

### Basique



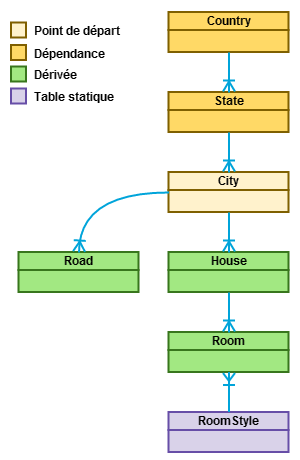
Pour être en mesure de créer un enregistrement City dans une autre base de données, il faut aller chercher les dépendances ou les clefs étrangères de la table de façon récursive. Dans notre cas, il faudra aller chercher la ligne State et Country en lien avec notre City.

### Basique dérivée



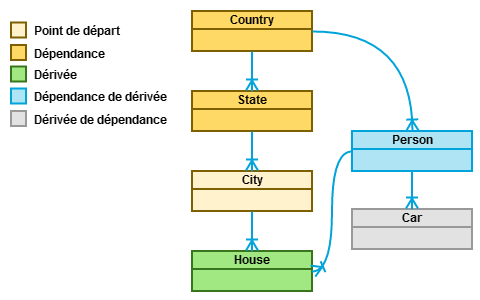
Généralement, lors de la copie d’une ligne, il serait de mise d’aller chercher les enregistrements dérivés. Il faut donc récupérer toutes les tables qui ont comme clef étrangère la clef primaire de la table de base puis sélectionner les lignes qui sont en relation et répéter le tout de façon récursive.

### Statique



Certaines tables contiennent des éléments unique et parfois utilisés par plus d’une table. Ces éléments ne doivent donc pas être dupliqués alors on en vérifie la présence avant d’aller plus loin. S’il existe, on utilise sa PK sinon on le crer. La création implique la récupération des dépendances. On peut spécifier les tables statique dans la configuration “staticTable”.

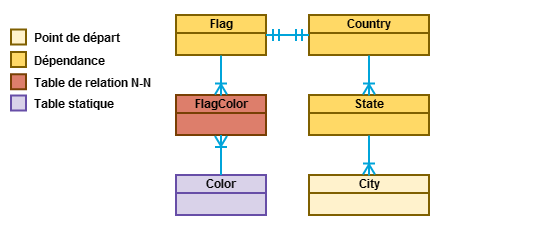
### Dépendance de dérivée



Pour l’enregistrement d’une dérivée, il faut aller chercher ses dépendances de façon récursive.

Pour chaque dépendance récupérée, il faut faire attention de ne pas sélectionner ses dérivées sous peine de ramenner la base de données au complet. S’il ont veut tout de même récupérer les tables dépendantes, on peut le faire en ajoutant les tables dans la configuration “derivativeTableAccess”. Par exemple : On récupère les House puis Person puis Country sans récupérer State et Car.

### Table de relation N-N



En temps normal, on ne récupère pas les dérivés, lorsqu’en mode de récupération des

dépendances mais il y a exeption à la règle lorsque la table est le résultat d’une relation plusieurs à plusieurs. La table doit se retrouver dans la configuration “manyToManyRelationshipsTable”.

# Prérequis

1. Création d’une classe « tableIdentifier » comportant :
   * ServerID as Int16 [XmlAttribute]
   * DatabaseName as String [XmlAttribute]
   * SchemaName as String [XmlAttribute]
   * TableName as String [XmlAttribute]
2. Création d’une classe « columnIdentifier : tableIdentifier » comportant :
   * ColumnName as String [XmlAttribute]
3. Création d’une classe « rowIdentifier » comportant :
   * TableIdentifier as ITableIdentifier
   * Columns as Dictionnary<string, value as object>
4. Création d’une classe “keyClonedDictionnary” représentant une table de correspondance entre les FK sous la forme “tableIdentifier / FK origine - tableIdentifier / FK destination”
   * Permet d’éliminer les boucles sans fin en gardant une trace de ce qui a déjà été fait.
   * Évite de créer des doublons si d'autres lignes font référence à la même donnée.
5. Création d’une classe “staticTableDictionnary“ représentant les tables qui ne doivent pas être copié sous la forme “tableIdentifier - active”. Hérite de IStaticTableDictionnary. Ex:  
   <StaticTables>

<server id="1">

<database name="root">

<schema name="dbo">

<table name="table1" active="true">

<table name="table2" active="true">

</schema>

</database>

</server>

</StaticTables>

* + Tables paramètres / valeurs

1. Création d’une classe “manyToManyRelationshipsTable” représentant les tables qui sont le réssultat des relations plusieurs à plusieurs. Ce sont des tables dérivées qui doivent être traitées comme des dépendences.  
   Ex:  
   <manyToManyRelationshipsTable>

<server id="1">

<database name="root">

<schema name="dbo">

<table name="table1" active="true">

<table name="table2" active="true">

</schema>

</database>

</server>

</manyToManyRelationshipsTable>

1. Création d’une classe “derivativeTableAccess” représentant les tables qui doivent dont les tables dérivées doivent être récupérés ou non.  
   Ex:  
   <DerivativeTableAccess>

<Server Id="1" xmlns="urn:DerivativeTableAccess">

<Database Name="db">

<Schema Name="dbo">

<Table Name="table1" Access="Denied" Active="true" />

<Table Name="table2" Access="Forced" Active="true" />

<Table Name="table2" Access="Inherited" Active="true" />

</Schema>

</Database>

</Server>

</DerivativeTableAccess>

1. Création d’une classe “extraForeignKeys” représentant des relations supplémentaires entre les tables sous la forme “(tableIdentifier / ColumnName) - (tableIdentifier / ColumnName), SQL, Mode=’Add | Modify | Remove’ ” :   
   Ex :   
   <extraForeignKeys>

<Server Id="1">

<Database Name="db">

<Schema Name="dbo">

<Table Name="personne">

<fk Name="" Mode="Add" serverIdDest="1" databaseDest="db" schemaDest="dbo" TableDest="personne" Active="True">

<column Name="idService" ColNameDest="idService" />

<column Name="idTransaction" ColNameDest="idTransaction" />

</fk>

<fk Name="pk" Mode="Modify" serverIdDest="1" databaseDest="db" schemaDest="dbo" TableDest="personne" Active="True">

<column Name="idPersonne" ColNameDest="idPersonneOlder" />

</fk>

<fk Name="fkBla" Mode="Remove" Active="True"/>

</table>

</Schema>

</Database>

</Server>

</extraForeignKeys>

* + Requis lorsque sous le modèle de l’héritage et que la clef primaire entre deux tables est implicitement la même.
  + Requis pour joindre deux bases de données
  + Pourrait être utilisé pour remplacer la jointure par défaut entre deux tables.
  + Permet d’aller chercher l’information de table dérivées ayant une liaison N-N.

1. Création d’une interface “IQueryProvider” : IDisposable qui indique les méthodes d’accès à une base de données.
   * Connection as IDbConnection
   * IsReadOnly as Bool
   * GetFK(tableIdentifier) as DataTable
   * GetLastInsertedPK() as Object
   * Select(rowIdentifier) as DataTable
   * Insert(tableIdentifier, datarow[]) as Bool
   * Update(rowIdentifier, datarow[]) as Bool
   * Delete(rowIdentifier) as Bool
2. Création des classes implémentant “IQueryProvider” : DBConnection pour chaque SGBD. Ex:
   * QueryProviderMySQL
   * QueryProviderSQLServer
   * QueryProviderOracle
3. Création d’une classe “QueryDispatcher : IQueryDispatcher, ICollection<IQueryProvider>” qui va récupérer la bonne implémentation de “IQueryProvider” pour exécuter la requête.
   * Dictionnary<serverId as Int16, value as IQueryProvider>
4. Création d’une classe “IInfoShemaTable”
   * Columns as Ilist<IInfoShemaColumn>
5. Création d’une classe “IInfoShemaColumn”
   * columnName as String
   * dataType as byte

# Configuration

## dc.config

Contient les configurations globales.

<?xml version="1.0"?>

<Configuration>

<StaticTables>

<server id="1">

<database name="root">

<schema name="dbo">

<table name="table1" active="true">

<table name="table2" active="true">

<table name="table3" active="true">

</schema>

</database>

</server>

</StaticTables>

<ConnectionStrings>

<add Id="1" Name="sql1" ProviderName="DataCloner.DataAccess.QueryProviderMySQL" ConnectionString="server=localhost;user id=root; password=cdxsza; database=mysql; pooling=false" />

</ConnectionStrings>

</Configuration>

# Logique du programme

Pour récupérer une PK non utilisé :

1. SELECT MAX(PK)+1 FROM MaTable
2. Vérifier si elle dépasse la capacité du type  
   SELECT DATA\_TYPE FROM INFORMATION\_SCHEMA.COLUMNS
   1. Si oui, vérifier s’il y a une PK non utilisé dans la lot.
      1. SELECT PK FROM MaTable
      2. Enumerable.Range(0, maxTypeSize)
      3. Faire une soustraction des deux listes
      4. Récupérer 1 clef qui n’est pas attribuée

Récupérer les types des colonnes d’une table :

* SQLLite : PRAGMA table\_info(*table-name*);
* MSSQL : SELECT DATA\_TYPE FROM INFORMATION\_SCHEMA.COLUMNS

<http://technet.microsoft.com/en-us/library/ms345522.aspx#_FAQ31>

Pour la récupération du shéma si la cache n’existe pas ou que le hash du fichier de config ne correspond pas à celui enregistré précédemment :

1. Pour chaque string de connection
2. Récupérer le shéma des tables de la BD par défaut
3. Récupérer les FK des tables
4. Modifier les FK avec extraForeignKeys
5. Précompiler les requêtes insert
6. Générer les objets de shéma
7. Sérialiser en binaire les objets
8. Enregistrer les objets groupés par base de données
9. Enregistrer le hash du fichier de config

Pour l’enregistrement dans un fichier :

1. Récupérer le shéma des tables du serveur d’origine qui ont des données à enregistrer.
2. Créer une BD SQLLite avec le shéma récupéré précédemment
   * Sans les pk / fk / contraintes / indexes
3. Enregistrer chacune des lignes

**Architecture du projet**

<http://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET>

<http://www.codeproject.com/Tips/662107/Understand-Tier-Architecture-in-Csharp>

<http://sharparchitecture.net/>

<http://programmers.stackexchange.com/questions/40394/how-do-you-organize-your-projects>

**Fichiers :**

[logical part name]

|-doc

|-lib

|- // any non-GAC-assemblies needed by the projects in the solution

|-src

|-Production

| |-Project1

| |-Project2

|-Tests

|-UnittestProject1

|-UnittestProject2

|-tools

|- // any tools needed for automated builds and such

// (NUnit, NCover, MSBuild tasks, ...)

**Espaces de nom :**

.ProjectNameDocuments

For large projects there are certain documents that need to be kept with

it. For this I actually create a seperate project or folder within the

solution to hold them.

.ProjectNameUnitTest

Unit testing always depends on the project some times it is just really

basic to catch edge cases and some times it is set up for full code

coverage. Recently have added graphical unit testing to the arsenal.

.ProjectNameInstaller

Some projects have specific installation requirements that need to be

handled at a project level.

.ProjectNameClassLibrary

If there is a need for web services, APIs, DLLs or such.

.ProjectNameScripts

This project holds the following types of scripts: SQL (Tables, procs, views),

SQL Data update scripts, VBScripts, etc.

.ProjectName

.DataRepository

Contains base data classes and database communication. Sometimes

also hold a directory that contains any SQL procs or other specific

code.

.DataClasses

Contains the base classes, structs, and enums that are used in the

project. These may be related to but not necessarily be connected to

the ones in the data repository.

.Services

Performs all CRUD actions with the Data, done in a way that the

repository can be changed out with no need to rewrite any higher

level code.

.Business

Performs any data calculations, business level data validation, does

most interaction with the Service layer.

.Helpers

These may be extensions on system items, standard validation tools,

regular expressions or custom built items.

.UserInterface

The user interface is built to display and manipulate the data.

UI Forms always get organized by functional unit namespace with an

additional folder for shard forms and one for custom controls.

**TODO**

Downcaster les types ex: columnIdentifier -> tableIdentifier

Utiliser memory cache : <http://www.codeproject.com/Articles/290935/Using-MemoryCache-in-Net>

==================================================================================================

SQLTraveler(byval riSource as rowIdentifier,

byval getDerivatives as bool,

byval shouldBeUnique as bool) as rowIdentifier

==================================================================================================

Initialiser une variable riRetour à null

Construire requête SELECT pour la riSource

Récupérer les lignes riSourceLines

Obtenir de la cache les FK de la table

Récupérer l'accès de la table aux tables dérivées (derivateTableAccess) currentTableAccess

Si currentTableAccess == denied

getDerivatives = false;

Sinon si currentTableAccess == forced

getDerivatives = true;

Si shouldBeUnique == true && lignes.count > 1

throw new exception();

Pour chaque ligne

Si la PK de la ligne est dans keyRelationshipDictionnary

Retourner la PK destination <--aaaaaaaaahhhhhh

Sinon

Dupliquer la ligne pour l'enregistrement de destination dans une variable lineDestination

Pour chaque contrainte de type FK

Si la FK est dans keyRelationshipDictionnary

Utiliser la clef depuis keyRelationshipDictionnary

Sinon

Si la table de destination de la FK est statique

Construire requête SELECT pour la ligne table destination

Récupérer la ligne

Si une ligne existe

Affecter la valeur des colonnes à celles de lineDestination

Sinon

Créer une rowIdentifier avec la valeur des colonnes de la contrainte

Appeler SQLTraveler(ri, false, true)

Affecter la valeur des colonnes à celles de lineDestination

Sinon

Créer une rowIdentifier avec la valeur des colonnes de la contrainte

Appeler SQLTraveler(ri, false, true)

Affecter la valeur des colonnes à celles de lineDestination

Créer la requête d'INSERT avec les valeurs de lineDestination

Enregistrer la ligne dans la table destination

Récupérer la PK nouvellement créée

Enregistrer la PK dans keyRelationshipDictionnary

Affecter riRetour avec la valeur des colonnes de la PK

Si getDerivatives == true

Obtenir de la cache les tables dérivées de riSource (contraintes FK qui pointent vers la PK de la table riSource)

Supprimer les tables qui sont dans derivateTableAccess avec l'access denied

Pour chaque table

Créer une rowIdentifier avec la valeur des colonnes de la contrainte (PK)

Appeler SQLTraveler(ri, getDerivatives)

=========================================================================

BuildCache()

=========================================================================

Création des FK par table

Obtenir le schéma des tables avec leur FK

Supprimer les FK reliées à des tables statiques