IUT de Bordeaux
Département Informatique

M2106 – Programmation et administration des bases de données

Mappage objet-relationnel (Transact-SQL)

Dictionnaire de données

Le **dictionnaire des données** permet, sous réserve de disposer des droits nécessaires, d'accéder aux informations concernant les bases de données hébergées par un serveur.

```
string conStr = "Provider=SQLOLEDB;Data Source=
      INFO-DORMEUR;Uid=ETD;Pwd=ETD";
OleDbConnection dbCon = new OleDbConnection(conStr);
dbCon.Open();
  // liste des bases
DataTable database =
  dbCon.GetOleDbSchemaTable(OleDbSchemaGuid.Catalogs,
      null);
  // liste des tables
DataTable tables =
  dbCon.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
      new object[] {null, null, null,"TABLE" });
  // etc. (plus d'information sur Moodle...)
```

Accès à la base en C#

Nous avons créé une application **Windows Form**, connectée à la base `MusiqueSQL`, et utilisant les tables `Musicien`, `Composer` et `Oeuvre`.

Pour effectuer le lien (**mappage**) entre les tables et notre application, nous avons créé deux classes, `Musicien` et `Oeuvre`, grâce auxquelles nous avons pu « convertir » des lignes de tables en objets...

```
while (reader.Read())
{
        int id = Convert.ToInt32(reader.GetInt32(0));
        string nom = reader.GetString(1);
        [...]
        Musicien m = new Musicien(id, nom, prénom);
        listBox1.Items.Add(m);
}
```

Entity Framework (EF)

Le module **Entity Framework** de **ADO.NET**, introduit avec la version 3.5 de **.Net**, permet d'automatiser ce mappage entre base de données relationnelle et application orientée objet.

Il est intégré comme un outil de Visual Studio : menu **Projet/Ajouter un nouvel élément/ADO.NET Entity Data model**, puis sélection de la base, des tables, des vues ou des procédures stockées.

Cela génère **une classe** par table sélectionnée, et une classe de gestion du mappage (fichier suffixé par `.Designer.cs`) associé à un fichier suffixé par `.edmx` qui décrit le schéma relationnel de la base.

LINQ (1)

LINQ (**Language Integrated Query**) est un composant **.NET** qui étend **C#** (et d'autres langages .NET) en offrant la possibilité d'utiliser une syntaxe de type **SQL** (**Select**) pour interroger toutes sortes de structures de données.

```
var results =  from c in SomeCollection
               where c.SomeProperty < 10
               select new {c.SomeProperty,
                           c.OtherProperty};

foreach (var result in results)
{
        Console.WriteLine(result);
}
```

LINQ (2)

La forme générale d'une requête **LINQ** est la suivante :

```
static void Main()
{
    // The Three Parts of a LINQ Query:
    // 1. Data source.
    int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };
    // 2. Query creation.
    // numQuery is an IEnumerable
    var numQuery = from num in numbers
                   where (num % 2) == 0
                   select num;
    // 3. Query execution.
    foreach (int num in numQuery)
    {
        Console.Write("{0,1} ", num);
    }
}
```

LINQ to SQL (accès à la base)

L'accès aux données de la table `Genre`, par exemple, se fera via une requête **Linq to SQL**, qui renvoie un objet de type `System.Data.Objects.ObjectQuery`.

Récupération d'un genre à partir de sa clé :

```
var gen = from g in musiqueSQL.Genre
          where g.Libellé_Abrégé == genre
          select g;
```

Récupération de tous les albums du genre ainsi sélectionné :

```
foreach (Album alb in gen.First().Album)
{
    string s = alb.Titre_Album;
}
```

Liste des musiciens : avant / après... (1)

Avant :

```
OleDbConnection connection = new OleDbConnection();
connection.ConnectionString = "....";
connection.Open();
string SQL =
   "Select Code_Musicien, Nom_Musicien from Musicien;";
OleDbCommand comm = new OleDbCommand(SQL, connection);
OleDbDataReader reader = comm.ExecuteReader();
List musiciens = new List();
while (reader.Read())
{
    Musicien a = new Musicien();
    a.Code_Musicien = reader.GetInt(0);
    a.Nom_Musicien = reader.GetString(1);
    a.Prénom_Musicien = reader.GetString(2);
    musiciens.Add(a);
}
```

Liste des musiciens : avant / après... (2)

Après :

```
MusiqueSQLEntities musiqueSQL = new MusiqueSQLEntities();
var musiciens  = from m in musiqueSQL.Musicien
                 orderby m.Nom_Musicien
                 select m;
foreach (Musicien m in musiciens) {
        Console.Write(m.Nom_Musicien + ", ");
}
Console.Read();
```

Avantage. La syntaxe est vérifiée dès la compilation, et non lors de l'accès au serveur, à l'exécution, pour une requête SQL de type string...

Opérations de mise à jour

Les **modifications** de la base de données (`INSERT`, `UPDATE` et `DELETE`) se feront par des modifications classiques des structures de données (modification d'un objet, création d'un nouvel objet ou suppression).

La **validation** des modifications effectuées se fera grâce à la méthode `SaveChanges` du gestionnaire de données :

```
musiqueSQL.SaveChanges();
```

Attention. *Cette méthode ne prend pas en charge l'ensemble des validations (contraintes référentielles, destruction en cascade…) qui devront être gérées par le programmeur lors de la construction ou de la suppression des objets. Exécution de type tout ou rien : toutes les opérations réussissent ou échouent...*

Et maintenant ?...

Proposez une nouvelle version de votre application (gestion des deux listbox) utilisant Entity Framework et LINQ...

Mise à jour / Proc. Stock. / Transactions (1)

```
using (var dbContextTransaction = musique.Database.BeginTransaction()) {
  try {
/*
var nom = new System.Data.SqlClient.SqlParameter("@Nom", "Ramet");
var prenom = new System.Data.SqlClient.SqlParameter("@Prénom", "Pierre");
var pays = new System.Data.SqlClient.SqlParameter("@NomPays", "France");
var login = new System.Data.SqlClient.SqlParameter("@Login", "pramet");
var pwd = new System.Data.SqlClient.SqlParameter("@Pwd", "pipo");
musique.Database.ExecuteSqlCommand(@"EXEC InsertAbonné @Nom , @Prénom, @NomPays, @Login
, @Pwd", nom, prenom, pays, login, pwd);
*/
    var res = musique.InsertAbonné("Ramet", "Pierre", "France", "pramet", "pipo");

//musique.Database.ExecuteSqlCommand(@"INSERT INTO Abonné (Nom_Abonné, Prénom_Abonné,
Login, Password, Code_Pays) VALUES ('Ramet', 'Pierre', 'pramet', 'pipo', 1)");
    Abonné a = new Abonné();
    a.Nom_Abonné = "Ramet";
    a.Prénom_Abonné = "Pierre";
    a.Login = "pramet";
    a.Password = "pipo";
    a.Code_Pays = 1; // France
    musique.Abonné.Add(a);
```

Mise à jour / Proc. Stock. / Transactions (2)

```
//musique.Database.ExecuteSqlCommand(@"UPDATE Abonné SET Password = 'newpwd' WHERE
Login LIKE '%ramet%'");
    a.Password = "newpwd";

//musique.Database.ExecuteSqlCommand(@"DELETE FROM Abonné WHERE Login LIKE '%ramet%'");
    var abonnes = from abo in musique.Abonné
                  where abo.Login.EndsWith("ramet")
                  select abo;
    foreach (Abonné ab in abonnes)
       musique.Abonné.Remove(ab);

    musique.SaveChanges();

    dbContextTransaction.Commit();
  }

  catch (Exception)
  {
    dbContextTransaction.Rollback();
  }

}
```