

Le Framework ADO.Net Entity

Alors que .Net 3.0 est sur le point de sortir, plusieurs équipes travaillent d'ores et déjà sur la version suivante, appelée pour l'instant, vNext. Les travaux portent sur de nombreux points : création d'applications pour Vista, pour Office 2007 et d'une manière générale faire tout ce qu'il faut pour faciliter la vie des développeurs. A ce titre, deux nouvelles technologies risquent de particulièrement retenir l'attention : le Framework ADO.Net Entity et LINQ (pour Language-Integrated Query).

Développer une application accédant à des données demande aujourd'hui la mise en œuvre de compétences multiples et d'outils variés n'offrant pas nécessairement un bon niveau d'intégration. Un développeur doit à la fois maîtriser :

- un langage de développement (C# ou VB).
- une couche d'accès aux données (ADO.Net ou les Enterprise Libraries).
- le SQL (voire plusieurs variantes si l'application accède à des données dans des bases d'éditeurs différents).
- Plusieurs outils (Visual Studio, SQL Server Management Studio, outils fournis avec les SGDB d'autres éditeurs).

Et bien que Visual Studio permette d'intégrer des outils d'accès aux données pour essayer d'unifier l'expérience utilisateur, des actions en théorie aussi simples que le débogage simultané de la partie code et de la partie SQL peuvent s'avérer pénible à l'usage. Or, en théorie, pour accéder aux données relatives à un client, le code suivant devrait être tout ce que le développeur devrait avoir à écrire :

```
using(MaConnection maBase = new MaConnection()) {
    // récupération de tous les clients membres du club
    // et livrant à l'adresse de facturation
    var clients = from client in maBase.Client
                  where client.EstMembre == true
                  && client Adresse.Type == 1
                  select client;

    foreach(Client client in clients) {
        // Traitement métier...
    }
}
```

Ce type de code expose ce qu'une solution de correspondance entre objets et base de données (ou mapping OR) doit pouvoir permettre. Le développeur peut alors se concentrer sur son code sans s'occuper de la plomberie sous-jacente, quitte à pouvoir ensuite affiner lorsque des problèmes d'optimisation se posent. La suite de cet article va explorer comment, avec le Framework Entity et LINQ, ce type de code peut être mis en œuvre.

Le Framework ADO.Net Entity

Pour illustrer tous les exemples à venir, cet article va s'appuyer sur une version simplifiée d'une base de données destinée à recueillir les commandes de clients d'un site Web. Cette base contient donc une liste de clients. Chaque client peut être membre du club des utilisateurs du site (avec des réductions à la clé par exemple). Chaque client peut disposer

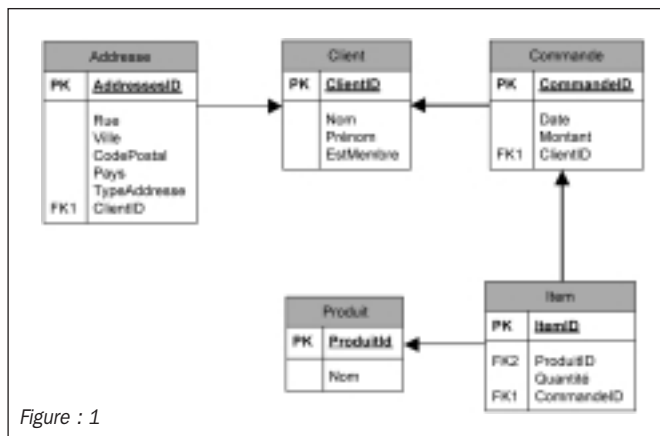


Figure : 1

de plusieurs adresses (facturation, livraison, etc). Et il a, bien entendu, la possibilité de passer plusieurs commandes :

Dans un développement classique, pour récupérer la liste de tous les clients ayant passé une commande depuis le début 2006 et qui sont membres du club, il faudrait commencer par écrire une requête SQL ressemblant à :

```
SELECT c.Nom, c.Prénom
FROM Client c
INNER JOIN Commande co ON co.ClientID = c.ClientID
WHERE c.EstMembre = 1 AND co.Date >= '2006-01-01'
```

Cette requête est encore assez simple : pour peu qu'il eût fallu aussi filtrer sur la nature des produits contenus dans la commande et le type d'adresse, plusieurs jointures auraient alors été nécessaires. Or la valeur ajoutée de ce type de requête est proche de zéro...

Le modèle de données Entity

Pour répondre à cette problématique, le Framework Entity introduit des éléments de modélisation permettant d'avoir une vue de plus haut niveau sur les structures de données. Ces éléments sont regroupés dans le modèle de données Entity (ou EDM pour Entity Data Model). En particulier, les concepts suivants sont définis :

- Entity, qui modélise une structure de données complexe. Les Entities peuvent être regroupées par Entity-Sets
 - Relationship, qui modèle une relation entre des Entities. Les Relationships peuvent être regroupées par Relationship-Sets.
- Ils permettent à l'EDM d'exprimer des notions comme :
- l'héritage (un ClientMembre est un type de Client).

- la gestion de type complexe (un Client a une adresse composée d'une rue, une ville, un code postal).

En pratique, le schéma logique peut alors être exprimé de la façon suivante (en se concentrant sur un client et ses commandes) : (Figure 2)

La requête SQL précédente peut alors être maintenant reformulée :

```
SELECT cm.Nom, cm.Prénom
FROM ClientMembre cm
WHERE cm.Commande.Date >= '2006-01-01'
```

Ici, la jointure est prise en charge par le Framework Entity grâce aux informations contenues dans le schéma EDM. A ce jour, la correspondance entre ce schéma et la structure de base de données sous-jacente se fait via l'utilisation de fichier XML. Des outils intégrés à Visual Studio seront fournis avec la version finalisée.

Le fournisseur de correspondance

Afin de pouvoir mettre en œuvre ces éléments dans une application .Net, un nouveau fournisseur de données a été créé. Ce dernier prend en paramètre les informations de correspondance et s'occupe alors de toute la plomberie. L'accès aux données depuis l'application peut alors se coder comme ceci :

```
using(MapConnection con = new
    MapConnection(Settings.Default.SiteWeb)) {
    con.Open();

    MapCommand cmd = con.CreateCommand();
    cmd.CommandText =
        "SELECT cm.Nom, cm.Prénom " +
        "FROM SiteWeb.SiteWebDB.ClientMembre AS cm " +
        "WHERE cm.Commande.Date >= @date";
    cmd.Parameters.AddWithValue("@date", DateDébut);

    DbDataReader r = cmd.ExecuteReader();
    while(r.Read()) {
        Console.WriteLine("{0}\t{1}", r["Prénom"], r["Nom"]);
    }
}
```

Ce code ne se distingue en rien de ce qu'aujourd'hui pourrait écrire un développeur utilisant ADO.Net 2.0. Mais à ce stade, l'accès aux données reste tabulaire. Une étape est encore nécessaire pour pouvoir récupérer des objets et travailler dessus.

La couche objet

Le Framework ADO.Net Entity est destiné à être accompagné de nombreux outils : création de schémas EDM, correspondance avec des structures de bases, etc.

Un de ces outils permet de générer des classes partielles à partir d'un schéma EDM : ainsi, le développeur peut ajouter du code métier (test de validité, calculs, autres) dans des fichiers distincts, sans craindre de perdre son code s'il génère de nouveau ces classes. Et elles permettent de réécrire le code précédent ainsi :

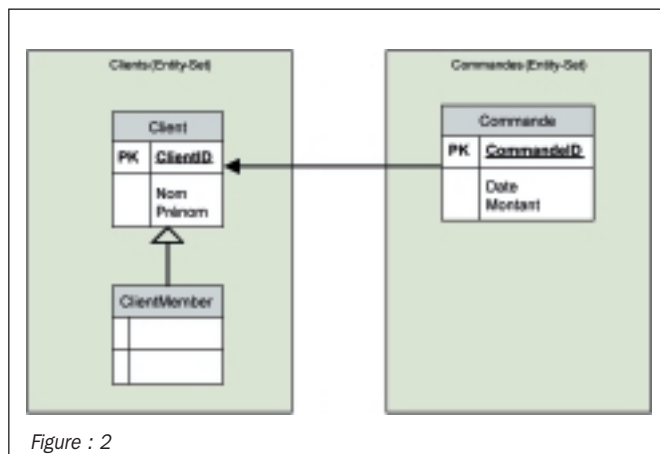


Figure : 2

```
using(MapConnection con = new
    MapConnection(Settings.Default.SiteWeb)) {
    con.Open();

   ObjectContext ctx = new ObjectContext(con);

    Query<ClientMembre> membres = ctx.GetQuery<ClientMembre>("SELECT cm.Nom, cm.Prénom " +
        "FROM SiteWeb.SiteWebDB.ClientMembre AS cm " +
        "WHERE cm.Commande.Date >= @date",
        new QueryParameter("date", DateDébut));

    foreach(ClientMembre membre in membres) {
        Console.WriteLine("{0}\t{1}", membre.Prénom, membre.Nom);
    }
}
```

Les objets ObjectContext et Query remplacent ici les objets Command et DataReader utilisés habituellement. Mais l'outil de génération de classes partielles fournit aussi une classe particulière, spécifique au schéma EDM utilisé, qui masque une partie de cette plomberie. Le code devient alors :

```
using(SiteWebDB sw = new
    SiteWebDB(Settings.Default.SiteWeb)) {

    Query<ClientMembre> membres = sw.GetQuery<ClientMembre>("SELECT cm.Nom, cm.Prénom " +
        "FROM SiteWeb.SiteWebDB.ClientMembre AS cm " +
        "WHERE cm.Commande.Date >= @date",
        new QueryParameter("date", DateDébut));

    foreach(ClientMembre membre in membres) {
        Console.WriteLine("{0}\t{1}", membre.Prénom, membre.Nom);
    }
}
```

Mais un point gênant reste : la nécessité de manipuler un langage de type SQL pour récupérer des objets .Net. Pour arriver au niveau du premier exemple de code de l'article, une dernière étape est nécessaire : l'utilisation de LINQ.

LINQ et Entities

Présentation de LINQ

LINQ était à l'origine un projet d'extension du langage C# devant permettre l'écriture de requêtes ensemblistes sans recourir à un autre langage. L'intégration de cette extension dans les différents langages du Framework permettra alors à ces langages de procéder à des sélections, des tris, des filtres sur différents types de données assimilables à une collection.

L'exemple de code suivant permet de récupérer dans un tableau d'entiers tous les éléments de valeur inférieure ou égale à 5, triés dans l'ordre croissant :

```
int[] numbers = new int[] {5, 7, 1, 4, 9, 3, 2, 6, 8};

var smallnumbers = from n in numbers
    where n <= 5
    orderby n
    select n;

foreach(var n in smallnumbers) {
    Console.WriteLine(n);
}
```

Comme indiqué précédemment, LINQ permet l'utilisation de requêtes ensemblistes sur de nombreux types de données (objets, XML). Il est aussi possible de requêter sur des DataSets en mémoire, sur des sources de données et sur des schémas EDM (exposés via le Framework Entity).

Utilisation conjointe avec SQL

Il s'agit probablement du mode d'accès aux données le plus simple. En effet, comme dans le cas du Framework Entity, un outil permet la génération de classes partielles à partir des structures de la base de données (tables, procédures stockées, vues et fonctions utilisateur).

Une des limitations majeures de ce mode d'utilisation est que, du fait de l'absence d'un schéma conceptuel intermédiaire, les classes générées reflètent directement la structure de la base : il n'est pas possible d'agglomérer deux tables dans une seule classe ou de générer des classes spécialisant une classe parent sur la base du contenu d'une colonne (comme la classe ClientMembre pour la classe Client dans les exemples précédents). Ce type de solution peut néanmoins parfaitement répondre aux besoins les plus simples.

Utilisation conjointe avec des DataSets

Dans certains cas, il peut être pertinent de cacher des données en mémoire, sans maintenir de connexion avec la base. Le Framework ADO.Net permet de satisfaire ce type de scénario depuis sa première version avec le type DataSet. Pour rappel, un DataSet permet la récupération en mémoire de données issues d'une base de données, tout en conservant une structure relationnelle (tables, relations, clés, index). Un langage de requête intégré permet même l'exécution de recherches simples.

Avec LINQ, il devient alors possible de travailler sur les données du DataSet :

```
DataSet ds = new DataSet();
ChargeClients(ds); // récupère les données depuis la base

DataTable clients = ds.Tables["Client"];

var query = from c in clients.ToQueryable()
    where c.Field<bool>("EstMembre") == true
    select new { ClientID = c.Field<Guid>("ClientID"),
        Nom = c.Field<string>("Nom"),
        Prénom = c.Field<string>("Prénom") };

foreach(var client in query) {
    Console.WriteLine("{0}\t{1}\t{2}",
        client.ClientID, client.Nom, client.Prénom);
}
```

Et chose intéressante : les requêtes sur jointure deviennent enfin possibles. Ainsi, pour rechercher tous les membres ayant passé une commande depuis début 2006, il faut écrire :

```
DataSet ds = new DataSet();
ChargeClients(ds); // récupère les données depuis la base

DataTable clients = ds.Tables["Client"];
DataTable commandes = ds.Tables["Commande"];

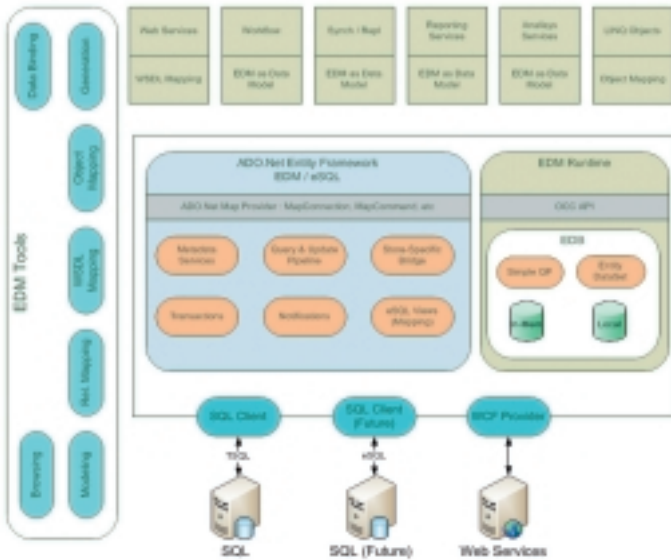
var query = from c in clients.ToQueryable()
    join co in commandes.ToQueryable()
    on co.Field<Guid>("ClientID")
    equals c.Field<Guid>("ClientID")
    where c.Field<bool>("EstMembre") == true
    && co.Field<DateTime>("Date") >= new DateTime(2006, 1, 1);
    select new { ClientID = c.Field<Guid>("ClientID"),
        Nom = c.Field<string>("Nom"),
        Prénom = c.Field<string>("Prénom") };

foreach(var client in query) {
    Console.WriteLine("{0}\t{1}\t{2}",
        client.ClientID, client.Nom, client.Prénom);
}
```

Mais ce type de code présente un inconvénient majeur : l'accès aux colonnes par leur nom rend le code moins lisible qu'il ne pourrait l'être et interdit toute vérification à la compilation. L'utilisation d'un DataSet typé permet de résoudre ce problème :

```
ClientDataSet ds = new ClientDataSet();
ChargeClients(ds); // récupère les données depuis la base

var query = from c in ds.Client
    join co in ds.Commande on co.ClientID equals c.ClientID
    where c.EstMembre == true
    && co.Date >= new DateTime(2006, 1, 1);
    select new { ClientID,
        Nom,
        Prénom };
}
```



```
foreach(var client in query) {
    Console.WriteLine("{0}\t{1}\t{2}",
        client.ClientID, client.Nom, client.Prénom);
}
```

Ce qui présente le double avantage d'être lisible et vérifiable à la compilation :

Utilisation conjointe avec le Framework Entity

Enfin, en utilisant LINQ au-dessus des classes générées à partir du schéma EDM, le code d'accès aux données peut enfin s'écrire de la manière suivante :

```
using(SiteWebDB sw = new
    SiteWebDB(Settings.Default.SiteWeb)) {
    var membres = from c in sw.ClientMembre
        // orderby c.Nom, c.Prénom, c.Commande.Date
        where c.Commande.Date > DateDébut
        select c;

    foreach(ClientMembre membre in membres) {
        Console.WriteLine("{0}\t{1}", membre.Prénom, membre.Nom);
    }
}
```

L'utilisation conjointe de LINQ et d'Entity a permis au final d'exprimer dans un seul langage, au sein d'un seul environnement, toutes les instructions nécessaires pour accéder en base aux données voulues. En résumé, ici :

- Entity a la charge de fournir le support pour un schéma conceptuel de données permettant une modélisation de plus haut niveau des structures en base. Il a aussi la charge de l'accès aux données (sélection, mise à jour, suppression). Enfin, il fournit des classes partielles permettant la manipulation des structures exposées par le schéma EDM dans un mode objet plutôt que tabulaire.
- LINQ a la charge de permettre l'expression de requêtes ensemblistes dans le langage utilisé par le développeur et non dans une variante de SQL. Il transforme ensuite ces requêtes en requêtes sur le schéma EDM sans que le développeur n'ait à intervenir sur ce processus.

Conclusion

De nombreuses questions restent néanmoins en suspens et demandent encore un peu de recul :

- LINQ to SQL et LINQ to Entities ne se recouvrent-ils pas ? L'absence d'une technique d'indirection (comme le schéma EDM d'Entity) semble limiter l'utilisation de LINQ to SQL aux cas les plus simples.
- Quel sera la place du langage SQL et des procédures stockées à l'avenir ? La tentation de ne plus utiliser que ce type de technologies peut être grande. Mais la problématique de l'optimisation de l'accès aux données, de la manipulation d'ensembles conséquents fait qu'un équilibre va devoir être trouvé entre la simplicité d'écriture et les besoins de performances.

Depuis le mois d'août, une version préliminaire est disponible sur MSDN pour tous ceux souhaitant commencer à tester ces technologies. Elle nécessite une version de Visual Studio, la version préliminaire de mai et une version de SQL Server (Express ou 2005) pour tester la persistance en base.

- La CTP de mai est téléchargeable sur <http://www.microsoft.com/downloads/details.aspx?familyid=1e902c21-340c-4d13-9f04-70eb5e3dceea&displaylang=en>
- La CTP d'août est téléchargeable sur <http://www.microsoft.com/downloads/details.aspx?familyid=b68f6f53-ec87-4122-b1c8-ee24a043bf72&displaylang=en>

Toutes les suggestions et remarques sont bien entendu les bienvenues :

■ Patrice Manac'h
Microsoft France



Offre Limitée

Abonnement 2 ans

Abonnez-vous pour 24 mois et recevez le livre « Métier Développeur - Kit de survie », de J-Chr. Arnulfo (Dunod, 272 pages). Offre réservée à la France métropolitaine.

90€ au lieu de : • le magazine seul, au numéro : 130,90 €
• Métier Développeur : 19,90 €

Economisez 60 Euros

Abonnez-vous sur www.programmez.com ou coupon page 59

