Veille technologique

LINQ to SQL

LINQ to SQL

Plan de la séance

- Introduction à LINQ to SQL
- Le DataContext
  - Obtenir la source de données
  - Écrire la requête
  - Exécuter la requête
- LINQ et les procédures stockées
- Application
- Conclusion

Introduction

- LINQ (Language Integrated Query) est une innovation dans Visual Studio 2008 et la version 3.5 du.NET Framework qui permet de rapprocher le monde des objets et le monde des données
- Dans LINQ to SQL le modèle relationnel d'une base de données est mappé en un modèle objet exprimé dans le langage de programmation (C#): ORM. Lorsque l'application est exécutée, LINQ to SQL convertit en SQL les requêtes intégrées au langage dans le modèle objet et les envoie à la base de données pour exécution.
- Lorsque la base de données renvoie les résultats, LINQ to SQL les convertit en objets que vous pouvez utiliser dans votre propre langage de programmation.
- Toutes les opérations de requête LINQ comportent trois actions distinctes :
    - Obtenir la source de données :
    - Créer la requête.
    - Exécuter la requête.

Obtenir la source de données: DataContext

La classes DataContext

- Un objet DataContext est est le conduit principal par lequel vous vous connectez à une base de données, récupérez des objets de celle-ci et soumettez des modifications.
- C'est la source de toutes les entités mappées sur une connexion de base de données. Il effectue le suivi des modifications que vous avez apportées à toutes les entités récupérées et gère un « cache d'identité » qui garantit que les entités récupérées plus d'une fois sont représentées à l'aide de la même instance d'objet
- Le DataContext est initialisé avec une chaine de connexion comme avec un SqlConnection

Obtenir la source de données: DataContext

Quelques constructeurs	Explications
DataContext(IDbConnection)	Initialise une nouvelle instance de la classe DataContext en référençant la connexion utilisée par le .NET Framework. La connexion est obtenue comme suit SqlConnection sqlconn= new SqlConnection(); sqlconn.ConnectionString = unechainedeconnexion;
DataContext(String)	Initialise une nouvelle instance de la classe DataContext en fournissant une chaine de connexion ou l'emplacement du fichier .mdf de la BD La chaine de connexion est sous la forme : string chaineConenxion = "Data Source=M-INFO-SY\\SQLEXPRESS2017;Initial Catalog=uneBD;User ID=user;Password=123456 le Data Source est le nom du serveur. Ce qui représente le nom de l'ordinateur suivi du nom de l'instance du serveur.

Obtenir la source de données: DataContext

- Exemple :

```
const string  chaine = "data source= PYACOUBS\\MSSQL2012; Initial Catalog =
PatocheBd; User Id = Patoche; password =remi2002";

private DataContext dcBd = new DataContext(chaine);
```

OU

```
private DataClasses1DataContext dcBd = new DataClasses1DataContext(chaine);
```

DataClasses1 est le nom de la classe LINQ.

Obtenir la source de données: DataContext

Méthodes importantes	Explications
CreateDatabase()	Créer une base de données sur le serveur. Le nom de la base de données est celui de chaine de connexion (si la BD n'existe pas)
ExecuteCommand(String, Object[])	Exécute une commande SQL directement sur la base de données : dcBd.ExecuteCommand("update questions set enonce ='qui suis-je ?' where idQuestion =12 "); dcBd.ExecuteCommand("create table joueurs(id int identity primary key, nom varchar(30))");
GetTable<TEntity>()	Retourne une collection d'objets d'un type particulier, où le type est défini par le paramètre TEntity. Table<categories> tablecategorie = dcBd.GetTable<categories>();
SubmitChanges()	Permet de mettre à jour la base de données. (officialiser une DML)
Dispose()	Libère toutes les ressources occupées par le DataContext
DeleteDatabase()	Supprime la base de données associée

Obtenir la source de données:

- La source de données peut être un tableau de données ou une partie des tables de la base de données.
- Lorsque la source de données utilise des tables de la base de données, celle-ci est obtenue comme suit :

```
Table <NomTableBD> nomsource = dcBd.GetTable <NomTableBD>();
```

- Exemple

```
Table <departements>  lesdepartements = dcBd.GetTable<departements>();
```

- La source de données est `lesdepartements` qui correspond à la table Departements de la base de données.

Créer la requête

- La requête spécifie les informations à récupérer de la source ou des sources de données. Elle peut également spécifier la manière dont ces informations doivent être triées, regroupées et mises en forme avant d'être retournées. Une requête est stockée dans une variable et initialisée avec une expression de requête. C# a introduit une nouvelle syntaxe pour simplifier l'écriture des requêtes.
- L'expression de requête contient trois clauses : from, where et select. Remarquez que l'ordre des clauses est inversé par rapport à l'ordre dans SQL. La clause from spécifie la source de données, la clause where applique le filtre et la clause select spécifie le type des éléments retournés.

Créer la requête

- **Exemple 1:**

```
var requ = from de in lesdepartements

select de.nomdepartement;
```

- **Exemple 2**

```
lesdepartements  = dcBd.GetTable<departements>();

lesemployes  = dcBd.GetTable<Employes>();

var jointure = from emp in lesemployes

               join de in lesdepartements   on emp.deptno equals de.deptno

               where de.nomdepartement == "inf"

              orderby emp.nom

              select emp.nom;
```

Exécuter la requête

Comme indiqué précédemment, la variable de requête elle-même stocke simplement les commandes de requête. L'exécution réelle de la requête est différée jusqu'à ce que vous itériez la variable de requête dans une instruction foreach. Ce concept, connu sous le nom d'exécution différée.

```
foreach (var resultat in jointure )
                {
              Console.WriteLine(resultat);
                }
```

Exécuter la requête

Pour les requêtes de mise à jour (INSERT, UPDATE et DELETE) elles se font au niveau programmation avec les méthodes :

- `InsertOnSubmit()` de la source de données; Uniquement pour l'insertion. Suivi de `SubmitChanges() du DataContext;`
- `SubmitChanges();` de la connexion (DataContext), pour les UPDATE
- `DeleteAllOnSubmit()` de la source de données, pour la suppression . Suivi de `SubmitChanges() du DataContext;`

Créer la requête

- Pour les requêtes de mise à jour (INSERT, UPDATE et DELETE) elles se font au niveau programmation avec les méthodes :
- `InsertOnSubmit()` de la source de données; Uniquement pour l'insertion. Suivi de `SubmitChanges() du DataContext;`
- `SubmitChanges();` de la connexion (DataContext), pour les UPDATE
- `DeleteAllOnSubmit()` de la source de données, pour la suppression . Suivi de `SubmitChanges() du DataContext;`

Créer et exécuter la requête

- Exemple: affichage (dans un DGV)

```
private void empDept()
    {
       lesdepartements = dcBd.GetTable<departements>();
       lesemployes = dcBd.GetTable<Employes>();
       var jointure = from emp in lesemployes
                      join de in lesdepartements on emp.deptno equals de.deptno
                      where de.nomdepartement == "informatique"
                      orderby emp.nom
                     select (emp.nom + de.nomDepartement);

       foreach (var listeEmp in jointure)
       {
         DGVQestions.Rows.Add(listeEmp);
       }
    }
```

Créer et exécuter la requête

- Exemple: Insertion

```
private void insertion()
      {
         lesemployes = dcBd.GetTable<Employes>();
          Employes emp1 = new Employes
              {
               empno = 222,
               nom = "Linq",
               prenom = "SQL"
               };

         lesemployes.InsertOnSubmit(emp1);
         sqlBd.SubmitChanges();
      }
```

Créer et exécuter la requête

- Exemple: Modification

```
private void modifier()
    {
       lesemployes = dcBd.GetTable<Employes>();
       var requete = from emp1 in lesemployes
                     where emp1.empno == 12
                     select emp1;
                    foreach (Employes  emp1 in requete)
                                {
                                 emp1.nom = "NouveauNom";
                                sqlBd.SubmitChanges();
                                }
    }
```

Créer et exécuter la requête

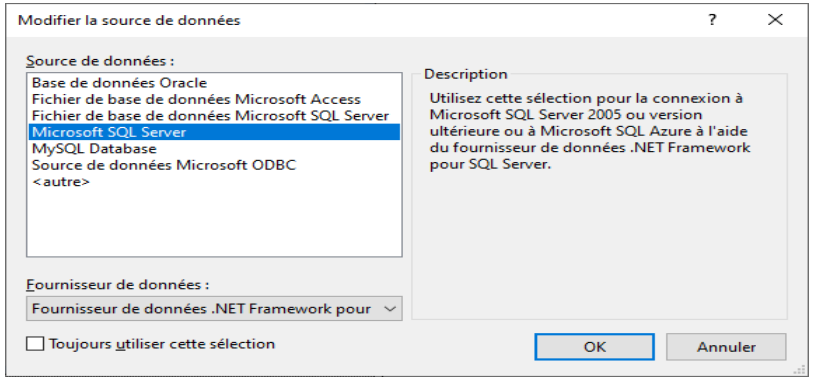
- Exemple: Suppression

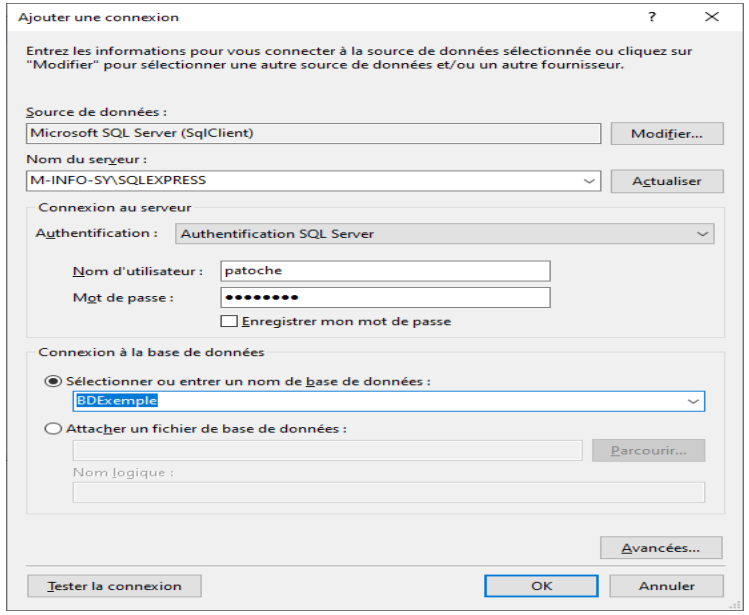
```
private void supprimer()
      {
         lesemployes = dcBd.GetTable<Employes>();
         var requetesup = from empsupp in lesemployes
                                  where empsupp.empno == 222
                                 select empsupp;
                                 foreach (var emp1 in requetesup)
                                          {
                                           lesemployes.DeleteAllOnSubmit(requetesup);
                                          sqlBd.SubmitChanges();
                                          }
      }
```

Créer un projet LINQ

- Démarrer un projet windowsForm, puis
- Établir une connexion au serveur.

Prenez le soin de choisir a bonne source de données(Microsoft SQL Server

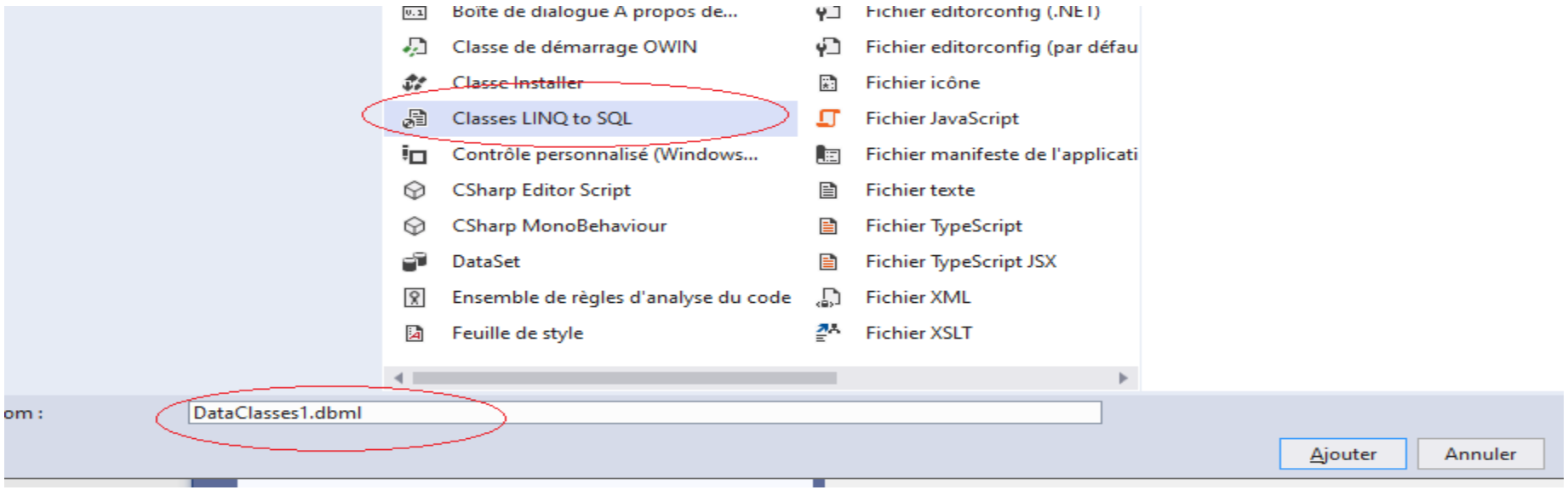




- Entrez le nom de votre serveur.
- Choisir Authentification **SQL Server**
- Entrez les informations de connexion
  - Nom d'utilisateur
  - Mot de passe
  - La base de données source.

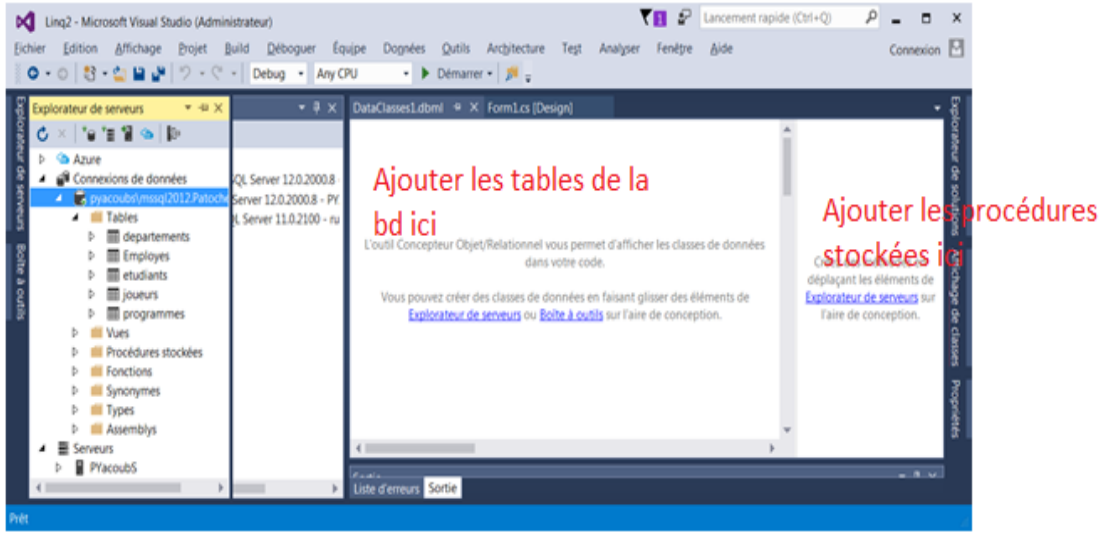
Créer un projet LINQ

- Ajouter classe LINQ to SQL à votre projet comme suit.
- Pour votre projet ajouter l'espace de nom: using System.Data.Linq;

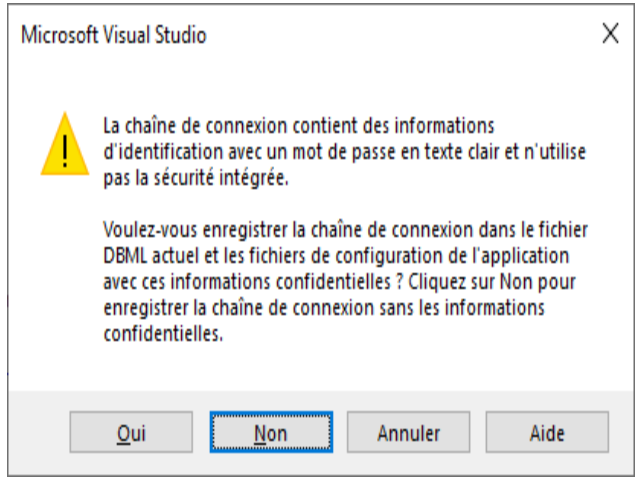


Créer un projet LINQ

- Ajouter les sources de données au projet : Dérouler votre serveur, puis glisser vos tables sur Explorateur de serveurs (voir figure suivante)

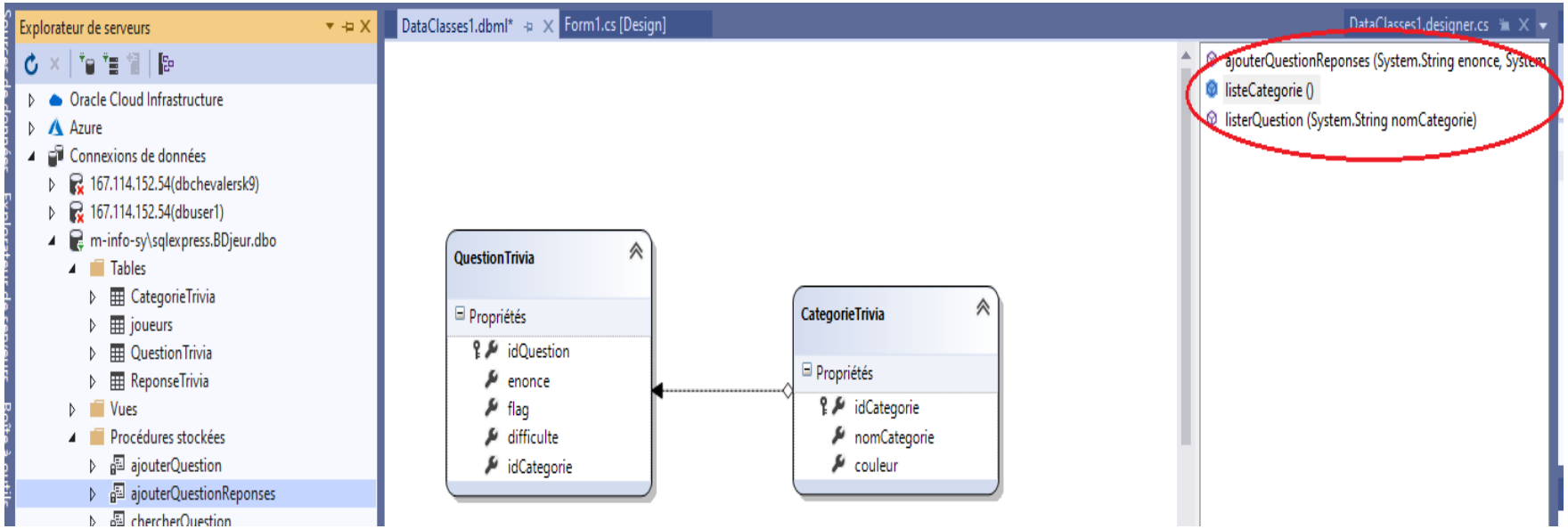


Le système va vous demander d'enregistrer la chaine de connexion dans App.config. Répondez par OUI.



Créer un projet LINQ

- Ajouter toutes les tables dont vous aurez besoin pour votre projet. Ajouter toutes les procédures stockées.



Application, ajouter une question, la procédure (insertion)

```
create procedure ajouterQuestionReponses (@enonce varchar(100),
 @idcategorie char(1),
 @difficulte char(1),
 @repA varchar(100),@estbonneA char(1),
 @repB varchar(100),@estbonneB char(1),
 @repC varchar(100),@estbonneC char(1),
 @repD varchar(100), @estbonneD char(1)) as
Begin
declare @idQuetion int;
begin try
begin transaction
insert into QuestionTrivia(enonce,difficulte,idCategorie) values (@enonce,@difficulte,@idcategorie);
select @idQuetion =@@IDENTITY;
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values(@repA, @estbonneA,@idQuetion);
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values(@repB, @estbonneB,@idQuetion);
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values(@repC, @estbonneC,@idQuetion);
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values(@repD, @estbonneD,@idQuetion);
commit;
end try
begin catch
rollback;
end catch
end;
```

Application, ajouter une question, le code C#

```
private void ajouerQuestion()
        {
            char ca = char.Parse(textCategorie.Text.ToString());
            char b1 = char.Parse(textEstbonne1.Text.ToString());
            char b2 = char.Parse(textEstbonne1.Text.ToString());
            char b3 = char.Parse(textEstbonne1.Text.ToString());
            char b4 = char.Parse(textEstbonne1.Text.ToString());
            char dif = char.Parse(textDifficulte.Text.ToString());

            dcBd.ajouterQuestionReponses(textEnonce.Text, ca, dif,
                                         textRep1.Text,  b1,
                                         textRep2.Text,  b2,
                                         textRep3.Text,  b3,
                                         textRep4.Text,  b4);
        }
```

Application, ajouter une question, le code C#

- Remarquez que dans le code C#, la procédure est considérée comme une méthode du d'un objet (dcBd) de la classe DataContext (DataClasses1DataContext).

- Dans ce cas, pour appeler la procédure, il suffit d'appeler la méthode en lui passant l'ensemble des paramètres.

- Si l'appel de la procédure dans SQL Server, l'ordre dans lequel les paramètres sont passés n'est pas important, pour la méthode du DataContext il faut les passer dans l'ordre

- Il est intéressant de voir à quel point, pour l'appel de procédures stockées, LINQ to SQL est vraiment plus facile que ADO.NET . Pensez à tous les paramètre qu'il faudra déclarer pour appeler la procédure.

Application, afficher la liste des catégories (Procédure , SELECT

```
CREATE OR ALTER  procedure  [dbo].[listeCategorie] as
BEGIN
select nomCategorie from categorieTrivia;
END

//----Le code C# correspondant qui affiche dans un comboBox--------
private void listerCategorie()
        {
            foreach (var listeCa in dcBd.listeCategorie())
            comboCategoTrivia.Items.Add(listeCa.nomCategorie);
            comboCategoTrivia.SelectedIndex = 0;
        }
```

Application, afficher la liste les questions d'une catégorie

```
CREATE OR ALTER   procedure [dbo].[listerQuestion](@nomCategorie varchar(30)) as
begin
        select enonce, difficulte, nomCategorie
        from QuestionTrivia Q inner join CategorieTrivia C
        on Q.idCategorie=c.idCategorie
        where @nomCategorie =nomCategorie;
end;
//----Le code C# correspondant qui affiche dans un DataGridView.--------
//----On utilise la propriété DataSource du DataGridView-------

private void comboCategoTrivia_SelectedIndexChanged(object sender, EventArgs e)
        {
            string nomCat = comboCategoTrivia.SelectedItem.ToString();
            DGVques2.DataSource = dcBd.listerQuestion(nomCat);
        }
```

Application, fonction table

```
CREATE OR ALTER function [dbo].[totalQuestioncategorie]() returns table as
return
 (
select count(*) as nbreQuestions, idcategorie
 from QuestionTrivia
 group by idCategorie
 );//----Le code C# correspondant qui affiche dans un DataGridView.--------
//----On utilise la propriété DataSource du DataGridView-------

 private void totalQuestionsCategorie()
        {
            DGVTotalQuestion.DataSource = dcBd.totalQuestioncategorie();
        }
```

Conclusion

- Il existe plusieurs façons d'exploiter une base de données relationnelle par .NET Framwork
  - ADO.NET
  - Entity Framework
  - LINQ to SQL
- Il ne faut pas conclure rapidement quelle technologie et meilleure par rapport à l'autre.
- Lorsque vous utilisez une petite base de données avec des procédures stockées, ou avec un nombre restreint de requêtes à la BD, LINQ to SQL semble avoir une bonne performance.
- Lorsque vous utilisez beaucoup de requête SQL, procédures stockées ou non, ADO.NET semble avoir une bonne performance.
- Comparé à ADO.NET, l'écriture du code LINQ to SQL, en procédures stockées offre un bel avantage.
- Si vous voulez mapper des objets entre votre base de données et votre code utilisez " Linq to SQL " ou " Entity Framework " **avec des procédures stockées.**

Présentation

- Sources:

- https://msdn.microsoft.com/fr-fr/library/system.data.linq.datacontext(v=vs.110).aspx
- https://docs.microsoft.com/en-us/dotnet/api/system.data.linq.datacontext?view=netframework-4.8
- https://www.codeproject.com/Articles/26431/Performance-Comparisons-LINQ-to-SQL-ADO-Csharp
- https://immobilis.developpez.com/tutoriels/dotnet/test-performance-acces-donnees-linq-vs-sql-vs-entity-framework/
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/linq-and-ado-net

Étude des besoins

CONCLUSION

QUESTIONS ??