(Leçon 04)
Le modèle - ADO.NET

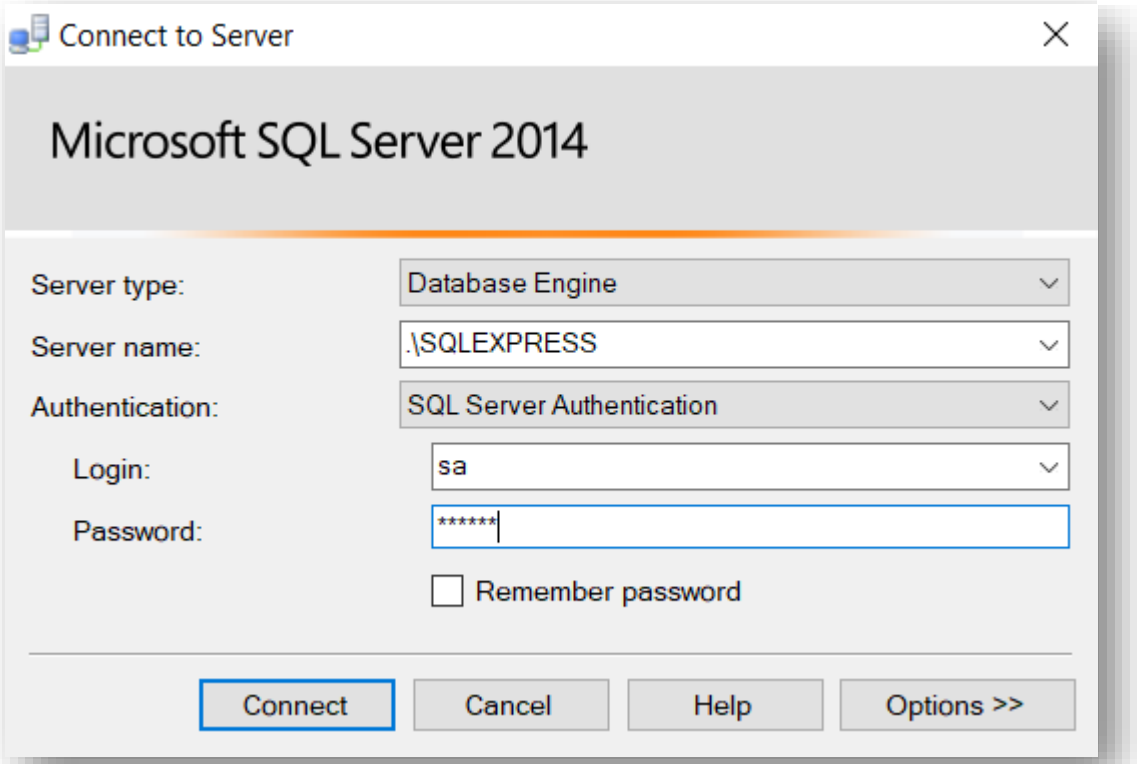
LOTFI DERBALI, PH.D
DERBALI.IT@GMAIL.COM
HTTPS://GITHUB.COM/DERBALI-IT

Contenu

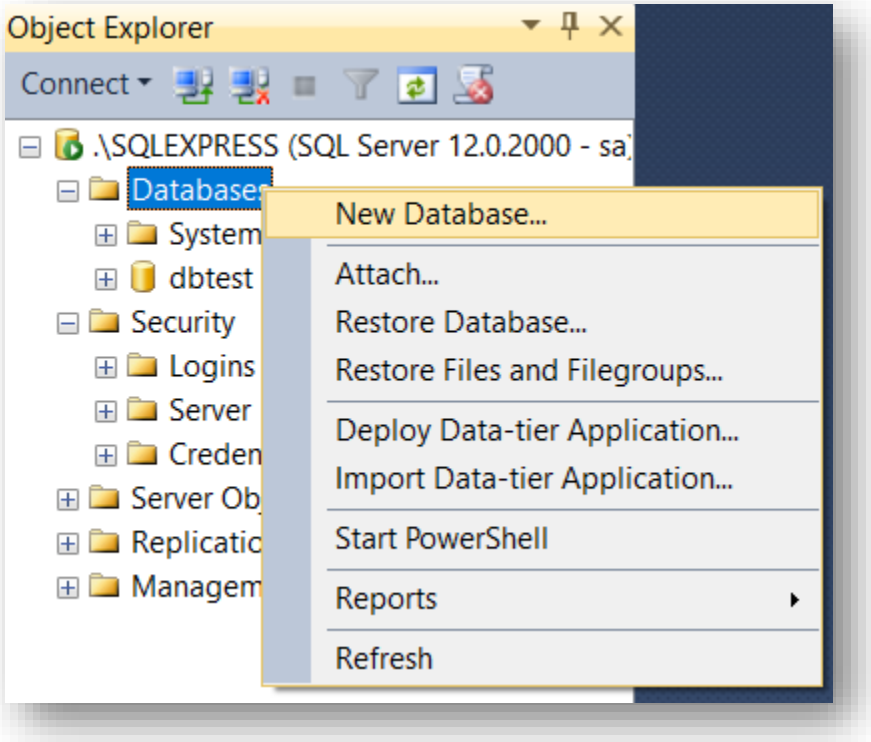
- Accès aux données d'une base de de données SQL Server
  - ADO.NET
  - Exécution d'une commande SQL Server à distance en C#.NET
  - Réception des données d'une base de données SQL Server en C#.NET
- Procédures stockées

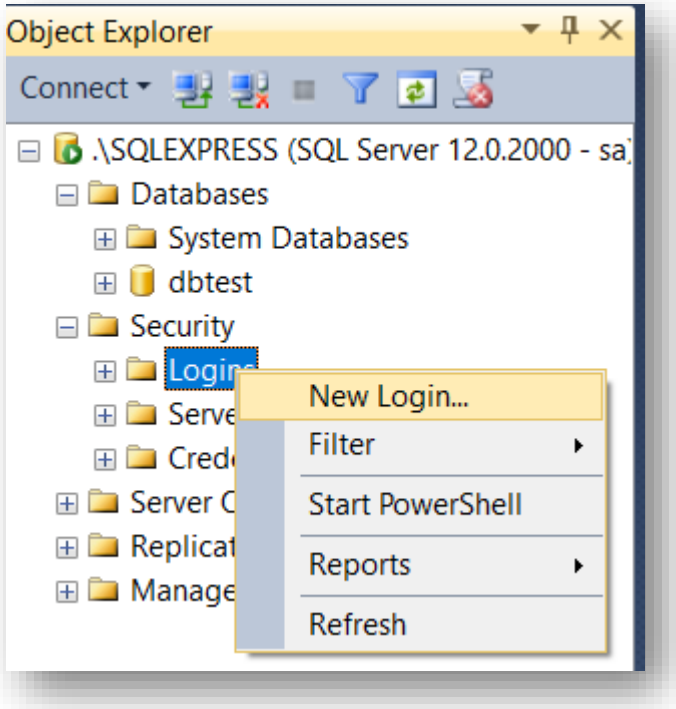
Rappel SQL Server

- Connexion à SQL Server Management Studio

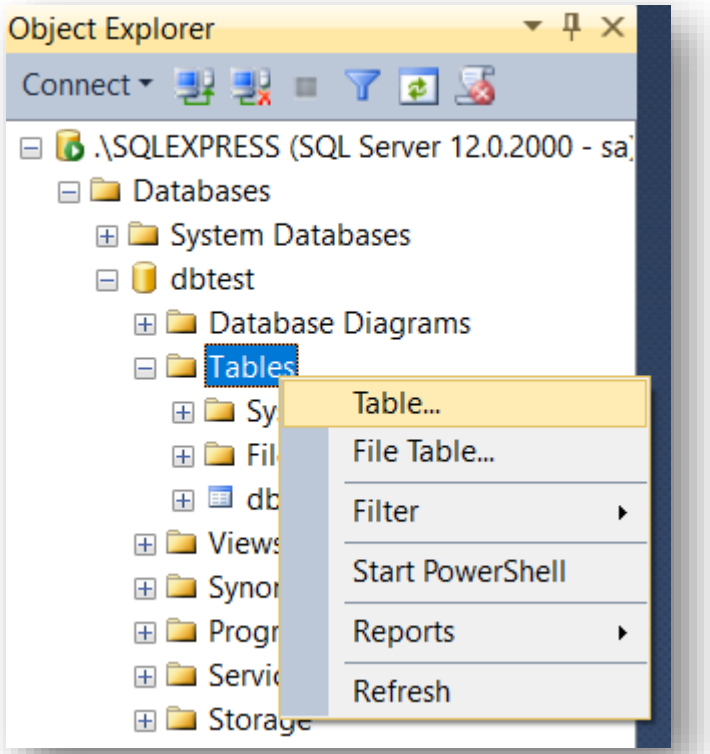


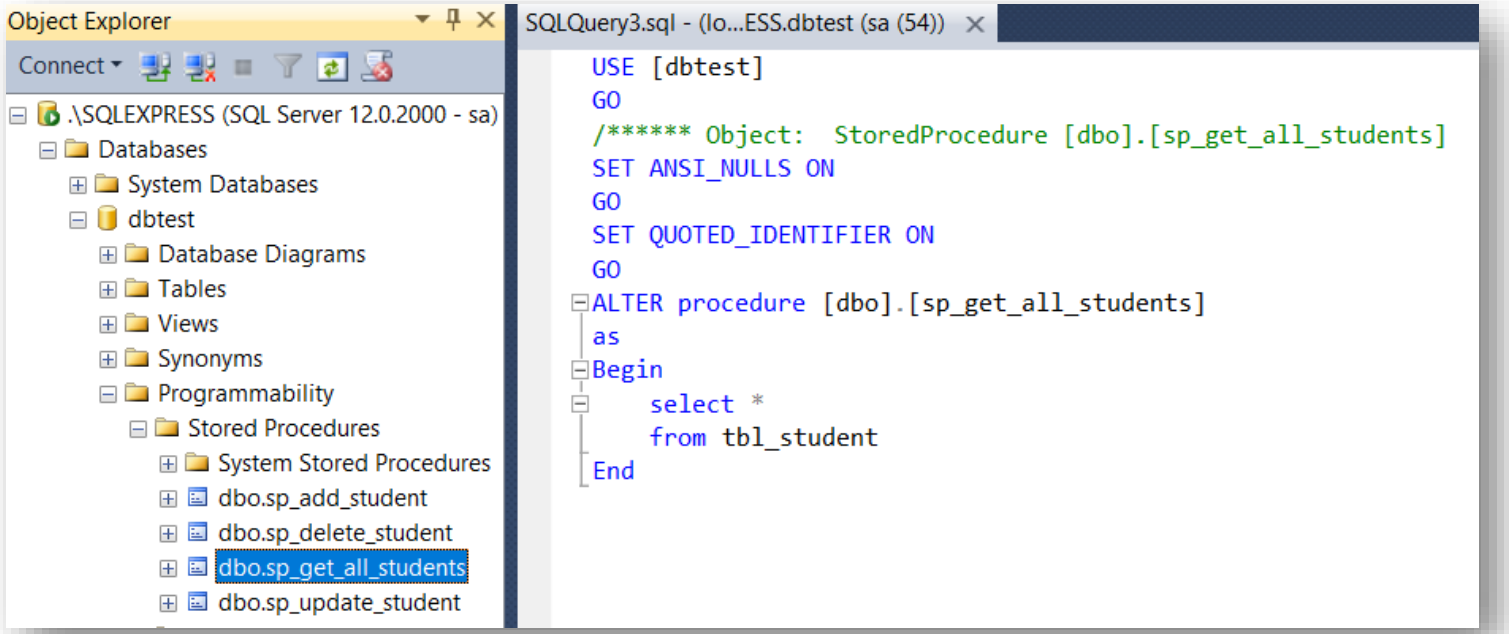
Nouvelle Base, nouvel utilisateur





Nouvelle table, nouvelle procédure stockée



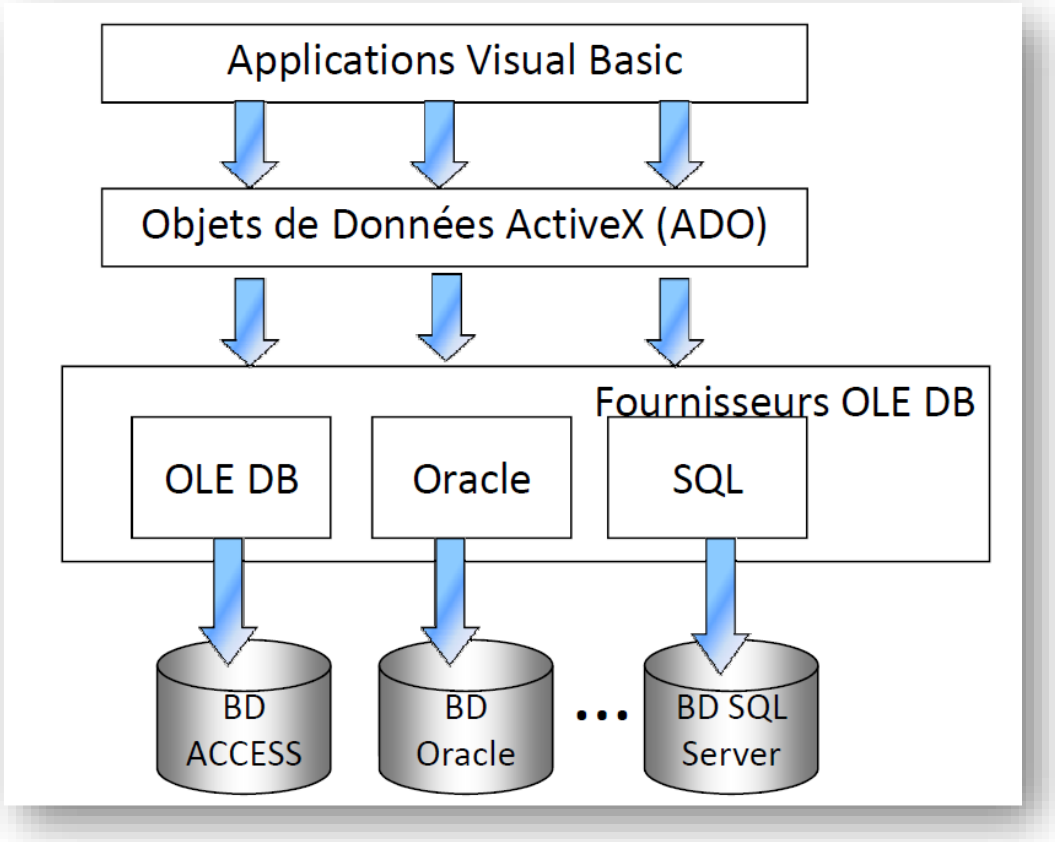


ADO.NET

- Accès à partir de C#.NET aux bases de données
  - Utilisation de la technologie ADO.NET (ActiveX Data Objects)
  - C'est la couche d'accès aux BDs

- ADO.NET
  - C'est un ensemble de classes, de méthodes et d'évènements permettant de récupérer et de manipuler des données

- ADO.NET permet un accès à différentes sources de données par l'intermédiaire de fournisseurs OLE DB (*Providers*)
  - Ces fournisseurs traduisent les requêtes dans le langage du système de base de données réel.
    - Par exemple le ASP.NET Core comprend le fournisseur de données SQL Server en mode natif via le namespace ***System.Data.SqlClient***
  - Ils permettent une manipulation identique quelque soit la source de données (SQL Server, MySQL, Oracle, MS Access, …)



Mode connecté et mode déconnecté

- Une connexion est une ressource qui est initialisée et utilisée par une application pour travailler avec une base de données.
- Une connexion avec une base de données peut prendre les états ouvert et fermé.
- Une application travaille en mode connecté si elle charge des données de la base au fur et à mesure de ses besoins. L'application reste alors connectée à la base.
- Une application travaille en mode déconnecté si elle charge des données de la base, qu'elle se déconnecte de la base, qu'elle exploite et modifie les données en mémoire, qu'elle se reconnecte à la base pour envoyer les modifications sur les données.

- Faiblesses du mode connecté / déconnecté:
  - La faiblesse du mode connecté est qu'il génère de très nombreux accès à la base de données et plus généralement, il génère de nombreux accès réseau si la base de données est séparée physiquement du reste de l'application.
  - La faiblesse du mode déconnecté est qu'il amène à consommer beaucoup de mémoire, puisqu'une partie de la base est récupérée en mémoire vive pour chaque appel client.

- Recommandation:
  - En ASP.NET Core, on recommande l'utilisation du mode connecté
    - Dans une page web, le cycle de vie du programme n'est que quelques secondes (ou minutes), les données n'ont donc pas besoin d'être stockées en mémoire

ADO.NET

La chaine de connexion (*connectionString*)

- un ensemble de paramètres qui définissent comment connecter une application à un SGBD

Mots-clés:

Keyword	Description
Data Source or **Server**	Specifies the name of a server.
Initial Catalog or **Database**	Specifies the name of a database on the server.
User ID or **uid**	Specifies the user name (for SQL Server Authentication).
Password or **pwd**	Specifies the user password (for SQL Server Authentication).

- Dans le projet ASP.NET Core,
  - Ajouter la *connectionString* dans le fichier **appsetting.json**

appsettings.json*

Schema: http://json.schemastore.org/appsettings

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "TestDatabase": "Server=.\\SQLEXPRESS;Database=Your database;User ID=valid user;Password=valid password;"
  }
}
```

Nom de connectionString

Seveur SQL

Nom de la base de données

Utilisateur

Mot de passe

Lecture de la *connectionString* en C#.NET

- Dans une classe DAL (Data Access Layer):

```
public class StudentDAL
    {
        public static IConfiguration Configuration { get; set; }

        //To Read ConnectionString from appsettings.json file
        private static string GetConnectionString()
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json");

            Configuration = builder.Build();
            string connectionString = Configuration["ConnectionStrings:TestDatabase"];
            return connectionString;

        }

        string connectionString = GetConnectionString();

       . . .
```

Exécution d'une commande SQL Server

- La classe qui réalise la connexion avec la base de données s'appelle SqlConnection
  - Pour se connecter à la base de données, il faut renseigner la chaîne de connexion (propriété *connectionString*), qui contient les informations nécessaires pour localiser et se connecter, puis ouvrir la connexion avec la méthode Open().
  - La classe commande SqlCommand permet d'exécuter des requêtes SQL, ou des procédures stockées avec ou sans paramètres. Les commandes exécutées peuvent ou non retourner des résultats.
  - Fermer la connexion avec la méthode Close().

Principales méthodes disponibles pour l'objet **SqlCommand**

ExecuteNonQuery()

- Exécute une instruction SQL sur la connexion et retourne le nombre de lignes affectées.
- Utilisée avec INSERT, UPDATE, DELETE

ExecuteReader()

- Génère une collection de type SqlDataReader
- Utilisée avec SELECT

ExecuteScalar()

- Exécute la requête et retourne la première colonne de la première ligne du jeu de résultats retourné par la requête. Les colonnes ou lignes supplémentaires sont ignorées.
- Utilisée avec les fonctions d'agrégation SQL (AVG, COUNT, SUM, MIN, MAX)

Obtenir des données avec l'objet SqlDataReader

Principales propriétés disponibles pour l'objet SqlDataReader

- FieldCount: Obtient le nombre de colonnes figurant dans la ligne actuelle.
- HasRows: Obtient une valeur qui indique si ce SqlDataReader contient une ou plusieurs lignes.
- RecordsAffected: Obtient une valeur indiquant si SqlDataReader contient une ou plusieurs lignes.

Principales méthodes disponibles pour l'objet SqlDataReader

- GetBoolean: Obtient la valeur de la colonne spécifiée sous la forme d'une valeur Boolean.
- GetDateTime: Obtient la valeur de la colonne spécifiée sous la forme d'un objet DateTime.
- GetDecimal: Obtient la valeur de la colonne spécifiée sous la forme d'un objet Decimal.
- GetDouble: Obtient la valeur de la colonne spécifiée sous la forme d'un nombre à virgule flottante double précision.
- GetFieldType: Obtient le type de données de la colonne spécifiée.
- GetFloat: Obtient la valeur de la colonne spécifiée sous la forme d'un nombre à virgule flottante simple précision.
- GetInt32: Obtient la valeur de la colonne spécifiée sous la forme d'un entier signé 32 bits.
- GetString: Obtient la valeur de la colonne spécifiée sous la forme d'une instance de String.
- Read: Avance le lecteur à l'enregistrement suivant dans un jeu de résultats.
- …

- Exemple: Récupérer les Students(Id, Name, Mark) à partir de la table tbl_student

```
public IEnumerable<Student> GetAllStudents()
        {
            List<Student> students = new List<Student>();

            using (SqlConnection con = new SqlConnection(connectionString))
            {
                string sqlStmt = "SELECT * FROM tbl_student";
                SqlCommand cmd = new SqlCommand(sqlStmt, con);

                con.Open();
                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Student student = new Student();
                    student.Id = rdr.GetInt32(0); //id
                    student.Name = rdr.GetString(1); //name
                    student.Mark = (double) rdr.GetDecimal(2); //mark

                    students.Add(student);
                }
                con.Close();
            }
            return students;
        }
```

Mettre à jour les données avec des requêtes SQL

- Utilisation des requêtes SQL INSERT, UPDATE ou DELETE
  - Appel de l'objet SqlCommand et de sa méthode ExecuteNonQuery()
  - La méthode ExecuteNonQuery() retourne le nombre de lignes traitées par la requête
    - Ceci permet de contrôler la bonne exécution de l'opération.

- Exemple: Ajouter un nouveau Student(Id, Name, Mark) à la table tbl_student

```
public void AddStudent(Student student)
        {
            using (SqlConnection con = new SqlConnection(connectionString))
            {
                string sqlStmt = "INSERT INTO tbl_student (name, mark) values(@name,@mark)";
                SqlCommand cmd = new SqlCommand(sqlStmt, con);

                cmd.Parameters.Add(new SqlParameter("@name", student.Name));
                cmd.Parameters.Add(new SqlParameter("@mark", student.Mark));

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
```

Utilisation des procédures stockées

- Au lieu d'utiliser une requête SQL, directement codée dans le source de l'application, il est possible d'utiliser une procédure stockée, créée préalablement dans la base.
  - La proximité d'un traitement dans une procédure stockée avec les données elles-mêmes est un avantage

Exemple: Récupérer les Students(Id, Name, Mark) à partir de la table tbl_student

```
public IEnumerable<Student> GetAllStudents()
        {
            List<Student> students = new List<Student>();

            using (SqlConnection con = new SqlConnection(connectionString))
            {
                SqlCommand cmd = new SqlCommand("sp_get_all_students", con);
                cmd.CommandType = CommandType.StoredProcedure;

                con.Open();
                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Student student = new Student();
                    student.Id = rdr.GetInt32(0); //id
                    student.Name = rdr.GetString(1); //name
                    student.Mark = (double) rdr.GetDecimal(2); //mark

                    students.Add(student);
                }
                con.Close();
            }
            return students;
        }
```

```
/****** Object:  StoredProcedure sp_get_all_students
CREATE procedure sp_get_all_students
as
Begin
    select *
    from tbl_student
End
```

Exemple: Ajouter un nouveau Student(Id, Name, Mark) à la table tbl_student

```
public void AddStudent(Student student)
        {
            using (SqlConnection con = new SqlConnection(connectionString))
            {
                SqlCommand cmd = new SqlCommand("sp_add_student", con);
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.Add(new SqlParameter("@name", student.Name));
                cmd.Parameters.Add(new SqlParameter("@mark", student.Mark));

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
```

```
/****** Object:  StoredProcedure sp_add_student
CREATE procedure sp_add_student
(
    @name VARCHAR(50),
    @mark DECIMAL(4,2)
)
as
Begin
    Insert into tbl_student(name, mark)
    Values (@name,@mark)
End
```