

# ADO.NET

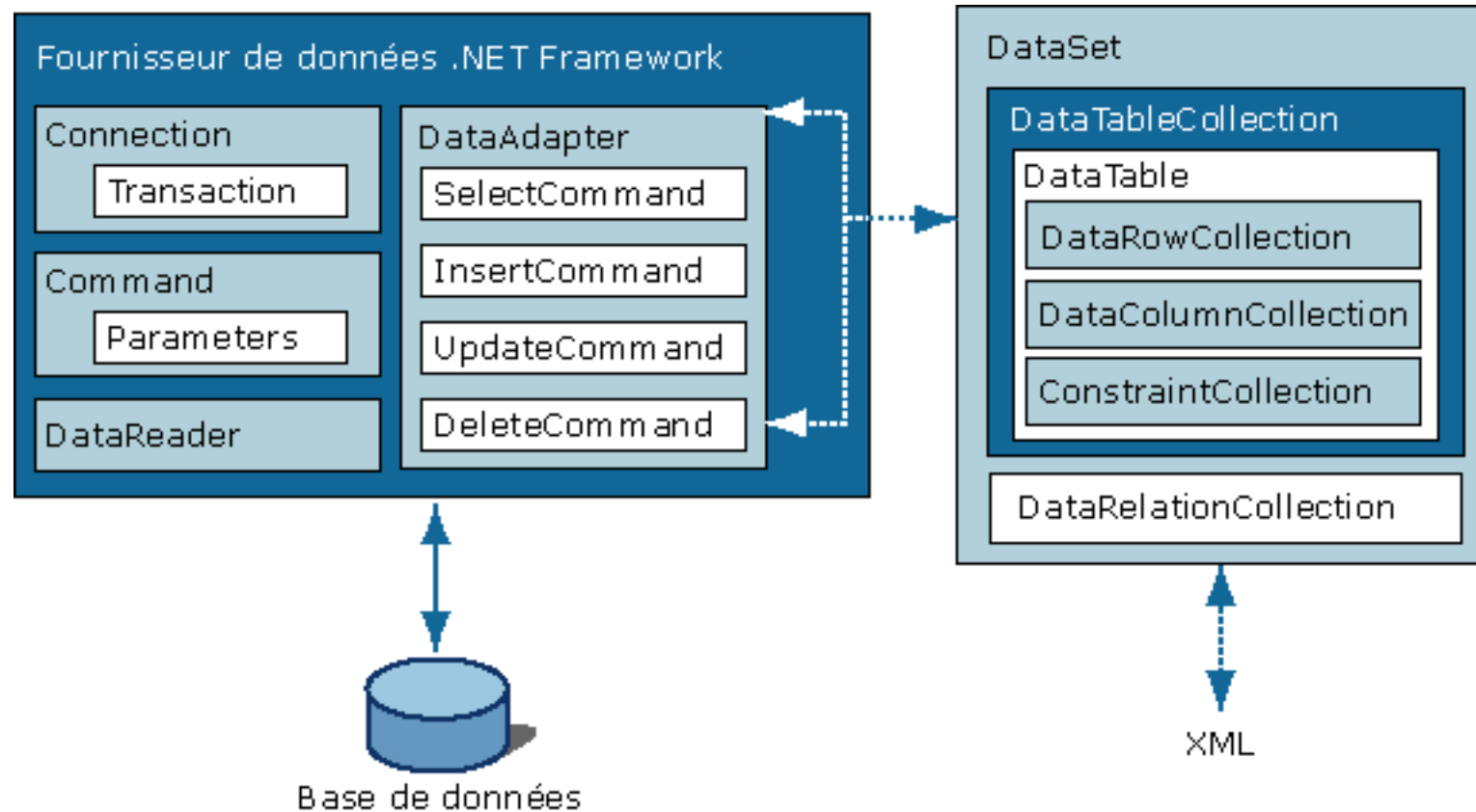
# Définition

- ADO.NET est une technologie de programmation utilisée pour accéder aux données à partir de sources de données relationnelles, telles que les bases de données Microsoft SQL Server, Oracle, MySQL, etc.
- ADO.NET est une partie de la plateforme .NET de Microsoft et fournit un ensemble de bibliothèques et de classes pour communiquer avec des bases de données.

# Composants d'ADO.NET

1. **Connecteur:** fournit une interface pour se connecter à une source de données et exécuter des requêtes sur cette source de données
2. **DataSet:** représente une collection de données qui peut être utilisée pour stocker et manipuler des données de manière indépendante de la source de données d'origine

# Schéma de l'architecture d'ADO.NET



# Connecteur SQL Server

- le connecteur pour SQL Server (SqlClient) utilise son propre protocole pour communiquer avec SQL Server
- Les classes du connecteur SQL Server sont situées dans l'espace de noms `System.Data.SqlClient`
- Pour installer le connecteur, utiliser le gestionnaire de paquet NuGet :
  - ❖ → outils → Gestionnaire de package NuGet → Gérer les paquets NuGet de la solution ... → Taper "*System.Data.SqlClient*"

# Classes principales du connecteur

Object	Description
Connection	Établit une connexion à une source de données spécifique
Command	Exécute une commande sur une source de données
DataReader	Lit un flux de données en avant uniquement (forward only) et en lecture seule à partir d'une source de données
DataAdapter	Remplit un DataSet et répercute les mises à jour dans la source de données

# La classe SqlConnection

SqlConnection représente une connexion à une base de données SQL Server

```
using (SqlConnection connection = new SqlConnection(connectionString)) {  
    connection.Open()  
    //effectuez vos opérations sur la base de données ici  
}
```

- Le bloc `using` libère automatiquement la connexion lorsque le code quitte le bloc
- La syntaxe d'une chaîne de connexion à SQL Server se trouve [ici](#)

# La classe SqlCommand

SqlCommand représente une instruction Transact-SQL ou une procédure stockée à exécuter par rapport à une base de données SQL Server

Méthode	utilisation
ExecuteReader	Execute une commande qui retourne des enregistrements
ExecuteNonQuery	Exécute des commandes d'insertions, modifications et suppressions et retourne le nombre de lignes affectées
ExecuteScalar	Récupère une valeur unique (un scalaire)



# Préparer une requête avec l'objet Command

- Instancier un objet SqlCommand permet de spécifier la requête SQL à exécuter sur la base de données
- Il est fortement recommandé d'ajouter des paramètres avec le caractère '@' dans une chaîne pour éviter les [injections SQL](#)

```
string queryString = "SELECT * FROM maTable WHERE nom=@nom;";  
SqlCommand command = new SqlCommand(queryString, connection);  
command.Parameters.AddWithValue("@nom", "John");  
// OU  
command.Parameters.Add(New SqlParameter("@nom", SqlDbType.VarChar));  
command.Parameters["@nom"].Value = "John";
```

# Lire des enregistrement avec DataReader

```
string queryString = "SELECT prenom, nom FROM salarie;";

using (SqlConnection connection = new SqlConnection(connectionString)) {
    SqlCommand command = new SqlCommand(queryString, connection);
    connection.Open();

    using(SqlDataReader reader = command.ExecuteReader()) {
        while(reader.Read()) {
            Console.WriteLine($"{reader(0)} {reader(1)}");
        }
    }
}
```

# Modifier des enregistrements avec ExecuteNonQuery

```
string queryString = "UPDATE salarie SET prenom=@prenom WHERE id=@id";

using (SqlConnection connection = new SqlConnection(connectionString)) {
    connection.Open();
    using (SqlCommand command = new SqlCommand(queryString, connection)) {
        command.Parameters.AddWithValue("@prenom", "Michel");
        command.Parameters.AddWithValue("@id", 3);
        int rowsAffected = command.ExecuteNonQuery();
        Console.WriteLine($"{rowsAffected} lignes affectées");
    }
}
```


# Récupérer une valeur avec ExecuteScalar

ExecuteScalar exécute la requête et retourne la première colonne de la première ligne dans le jeu de résultats retourné par la requête

```
string queryString = "SELECT MAX(salaire) FROM salarie";

using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    using (SqlCommand command = new SqlCommand(queryString, connection))
    {
        int salaireMax = (int)command.ExecuteScalar();
        Console.WriteLine($"Le salaire max est de {salaireMax} euros");
    }
}
```

# Le mécanisme de transaction

- Les transactions dans ADO.NET servent à lier plusieurs tâches ensemble de sorte qu'elles s'exécutent comme une seule unité de travail
- Si une requête d'une transaction échoue, les mises à jour peuvent être annulées
-  Les transactions impliquant plusieurs ressources peuvent réduire l'accès concurrentiel si des verrous sont conservés trop longtemps. Par conséquent, réduisez autant que possible les transactions

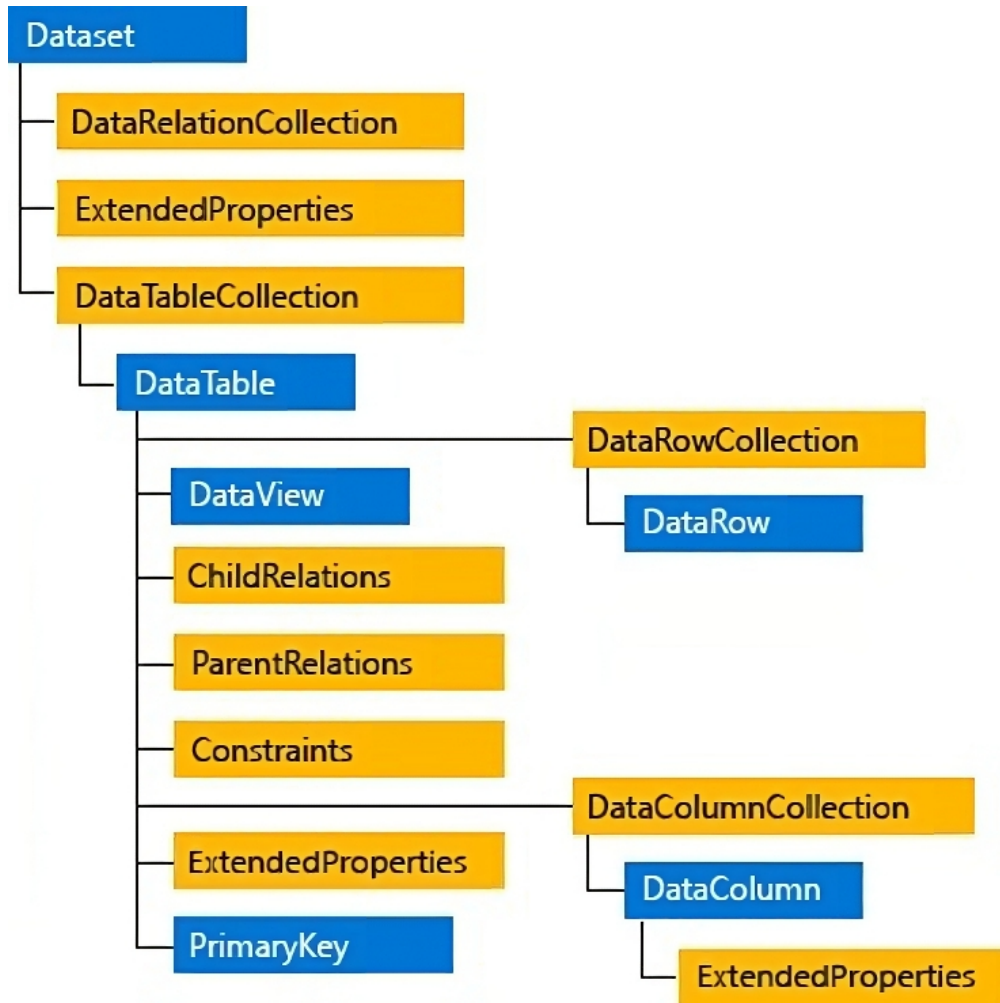
# Réaliser une transaction

```
string query = "DELETE FROM salarie WHERE id = 3;";
using (SqlConnection connection = new SqlConnection(connectionString)) {
    connection.Open();
    using (SqlTransaction tran = connection.BeginTransaction()) {
        using (SqlCommand command = new SqlCommand(query, connection, tran)) {
            if (command.ExecuteNonQuery() == 1) {
                tran.Commit(); // Validation de la transaction
            }
            else {
                tran.Rollback(); // Annulation de la transaction
            }
        }
    }
}
```

# Datasets ADO.NET

- Le DataSet ADO.NET est conçu pour un accès aux données indépendant de toute source de données
- Le DataSet contient une collection d'un ou plusieurs objets **DataTable** constitués de lignes et de colonnes de données, ainsi que des informations concernant les contraintes de clé primaire, de clé étrangère et des informations relationnelles sur les données contenues dans les objets DataTable

# Schéma descriptif d'un DataSet





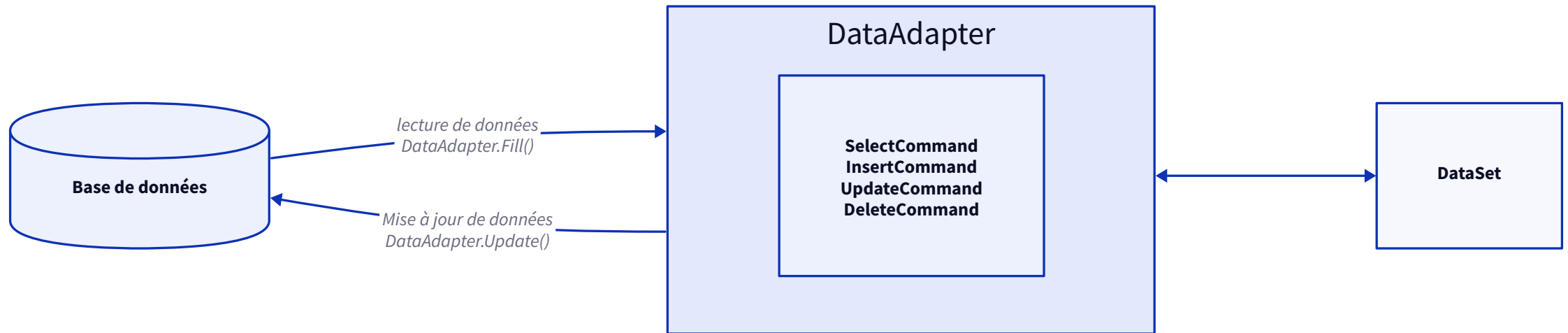
# Comment utiliser un DataSet

Il existe 3 méthodes pour initialiser un DataSet

1. Par programmation des objets : DataTable, DataRelation et Constraint
2. Par remplissage d'un DataSet à l'aide d'un objet DataAdapter
3. Par remplissage à partir d'un flux ou d'un document XML

Pour en savoir plus : [Documentation](#)

# Fonctionnement du DataAdapter



# Remplir un DataSet avec un DataAdapter

```
string query = "SELECT salarie_id, prenom, nom FROM salarie;";
using (SqlConnection connection = new SqlConnection(connectionString)) {
    SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

    DataSet salarie = new DataSet();
    adapter.Fill(salarie, "salarie");

    foreach (DataRow row in salarie.Tables["salarie"].Rows) {
        Console.WriteLine($"{row["salarie_id"]} prenom: {row["prenom"]}");
    }
}
```

# Aller plus loin avec Entity Framework

- Nous venons de voir que ADO.NET fournit un ensemble d'API pour accéder aux données à partir de sources de données relationnelles
- Cependant, pour les développeurs qui cherchent à simplifier le développement de leurs applications basées sur des données, il existe une solution plus évoluée : Entity Framework
- Entity Framework est un outil de mapping objet-relationnel (ORM) qui permet de travailler avec des données relationnelles en utilisant des objets et des méthodes plutôt que des requêtes SQL

**Merci pour votre attention**

**Des questions ?**