Utiliser ADO.NET en mode connecté

Le mode connecté est le mode originel d'utilisation des bases de données. Il permet de rester connecté en permanence à la source de données, ce qui présente certains avantages :

- Il est très simple d'utilisation. En effet, dans ce mode, la connexion à la source de données est créée au démarrage de l'application et détruite lorsque l'utilisateur la quitte.
- Il permet d'avoir en permanence des données à jour.
- Il permet de gérer plus simplement les accès concurrentiels. Les utilisateurs étant connectés en permanence, il est plus simple de déterminer lequel utilise les données.

Le mode connecté pose tout de même quelques problèmes non négligeables :

- Il impose le maintien d'une connexion réseau permanente entre le client et le serveur de base de données. À l'heure de la mobilité, il est particulièrement délicat d'imposer cela à un utilisateur.
- Le serveur de bases de données doit gérer de multiples connexions permanentes simultanément, ce qui peut induire une utilisation excessive de ressources.

1. Connexion à une base de données

Avant toute exécution de requête, il est nécessaire d'ouvrir une connexion vers le serveur de base de données. Cette connexion est créée à l'aide de la classe `SqlConnection`. Lors de l'instanciation d'une connexion, il est nécessaire d'initialiser certaines valeurs relatives à la base de données. Celles-ci sont spécifiées au travers d'une chaîne de connexion.

a. Chaînes de connexion

Une chaîne de connexion est une chaîne de caractères contenant les informations nécessaires à l'établissement d'une connexion avec une base de données. Ces informations sont spécifiées sous la forme de couples mot-clé/valeur séparés par des points-virgules, dans lesquels le mot-clé est un nom de paramètre reconnu par le fournisseur de données.

Une analyse de la chaîne est effectuée lors de son affectation à la propriété `ConnectionString` d'un objet `SqlConnection`. Au cours de cette analyse, les différentes valeurs contenues sont extraites et affectées aux différentes propriétés de la connexion. En cas d'erreur pendant cette étape d'analyse, une exception est levée et les propriétés de la connexion restent inchangées.

Les mots-clés suivants sont disponibles pour une chaîne de connexion :

Data Source

Nom ou adresse réseau du serveur vers lequel la connexion doit être établie. Il est possible de spécifier un numéro de port à sa suite en séparant ces valeurs par une virgule. La valeur par défaut du numéro de port est 1433.

```
Data Source=SRVSQL
```

Initial Catalog

Nom de la base de données sur laquelle doit s'effectuer la connexion.

```
Initial Catalog=Northwind
```

User Id

Nom d'utilisateur SQL Server à utiliser pour la connexion. Toute requête exécutée au travers de la connexion sera soumise aux droits d'accès de ce compte utilisateur.

```
User Id=sqluser
```

Password

Mot de passe associé au compte utilisateur SQL Server utilisé.

```
Password=utilisateur7
```

Integrated Security

Valeur booléenne donnant la possibilité d'utiliser l'authentification Windows courante pour la connexion.

```
Integrated Security=True
```

Connect Timeout

Durée (en secondes) pendant laquelle l'application attend l'établissement de la connexion. En cas de dépassement de cette durée, une exception est levée. Par défaut, cette durée est de 15 secondes.

```
Connect Timeout=30
```

Min Pool Size

Nombre minimum de connexions à conserver dans le pool de connexions.

```
Min Pool Size=20
```

Max Pool Size

Nombre maximum de connexions à conserver dans le pool de connexions.

```
Max Pool Size=70
```

La notion de pool de connexions est traitée en détail un peu plus loin dans ce chapitre.

Ainsi, pour établir une connexion au serveur **SRVSQL**, sur la base **Northwind** avec le nom d'utilisateur **sqluser** et le mot de passe **utilisateur7**, on utilisera la chaîne de connexion suivante :

```
string chaine = "Data Source=SRVSQL;Initial
Catalog=Northwind;User Id=sqluser;Pwd=utilisateur7";
```

Il peut être délicat de maîtriser la totalité des paramètres utilisables dans une chaîne de connexion. La classe `SqlConnectionStringBuilder` a pour but de simplifier la création d'une chaîne de connexion. Ce type possède plusieurs propriétés qui correspondent chacune à un paramètre utilisable dans une chaîne de connexion. Ces propriétés sont documentées de manière à comprendre aisément leur intérêt. Ainsi, il n'est pas nécessaire de connaître parfaitement chaque mot-clé utilisable pour obtenir une chaîne de connexion fonctionnelle.

```
SqlConnectionStringBuilder builder = new
SqlConnectionStringBuilder();
builder.DataSource = "SRVSQL";
builder.InitialCatalog = "Northwind";
builder.UserID = "sqluser";
builder.Password = "utilisateur7";

Console.WriteLine("Chaîne de connexion : {0}",
builder.ConnectionString);
```

Le résultat de l'exécution de ce code est le suivant :

```
Chaîne de connexion : Data Source=SRVSQL;Initial
Catalog=Northwind;User ID=sqluser;Password=utilisateur7
```

b. Pools de connexions

Les pools de connexions permettent d'améliorer les performances d'une application en évitant la création de connexions lorsque ceci est possible.

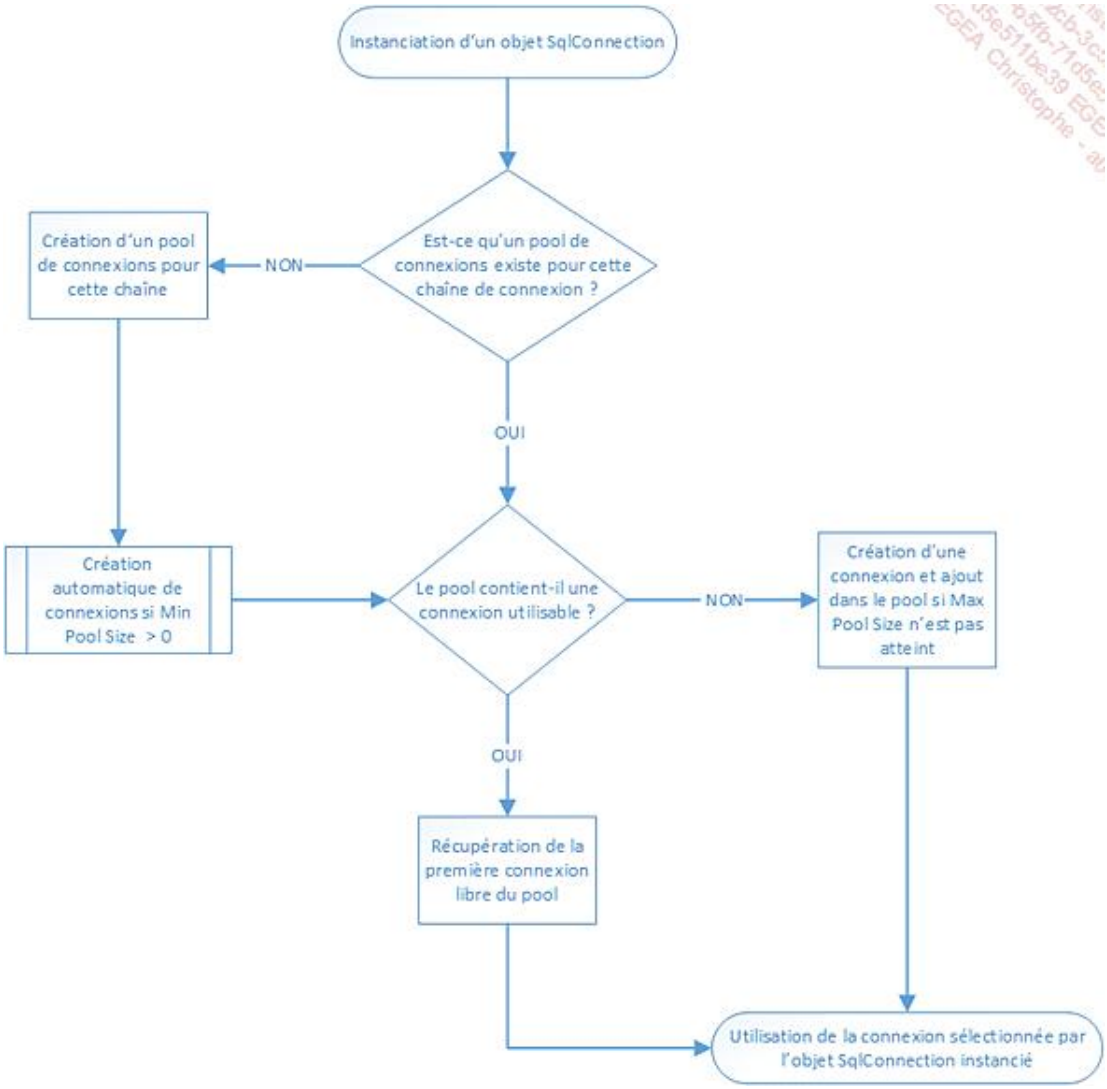
L'établissement d'une connexion est coûteux en termes de ressources, tant du côté client que du côté serveur. Le principe du pool de connexions est la mise en cache de connexions pour une réutilisation maximale. Dans ce but, un pool permet de conserver une certaine quantité de connexions utilisant la même chaîne de connexion. Lorsque l'application initie une connexion à une base de données, la première connexion disponible dans le pool associé à la chaîne de connexion spécifiée est automatiquement utilisée.

Les pools ainsi créés existent tant que l'application est en cours d'exécution.

Pour chacun des pools, il est possible de spécifier, par l'intermédiaire des propriétés `Min Pool Size` et `Max Pool Size` de la chaîne de connexion, les nombres minimum et maximum de connexions qu'il doit contenir.

Si le nombre maximal de connexions dans le pool est atteint, et qu'aucune connexion n'est disponible dans le pool, la demande sera mise en attente jusqu'à ce qu'une connexion soit disponible. Une connexion est remise à la disposition du pool lorsqu'elle est fermée explicitement dans le code à l'aide de la méthode `Close`, ou lorsque la méthode `Dispose` de la connexion est appelée. Pour assurer le recyclage des connexions, il est donc important de les fermer lorsqu'elles ne sont plus utilisées.

Ci-dessous une représentation schématique de l'utilisation d'un pool de connexions.



c. Gestion de la connexion

Établissement de la connexion

Pour créer une connexion à une base de données, il est nécessaire d'instancier un objet `SqlConnection` et d'initialiser sa chaîne de connexion, ce qui peut être fait de deux façons :

- Via le paramètre `connectionString` du constructeur de la classe `SqlConnection`.
- Via l'affectation d'une valeur à la propriété `ConnectionString` de l'objet `SqlConnection`. Cette affectation ne peut se faire que lorsque la connexion est fermée.

```
string chaineDeConnexion = "Data Source=SRVSQL;Initial
```

```
Catalog=Northwind;User ID=sqluser;Password=utilisateur7";

SqlConnection connexion;

//Initialisation à partir du constructeur
connexion = new SqlConnection(chaineDeConnexion);

//Ou initialisation à partir de la propriété
connexion = new SqlConnection();
connexion.ConnectionString = chaineDeConnexion;
```

Une fois la destination de la connexion définie, il faut appeler la méthode `Open` de l'objet `connexion` pour ouvrir la connexion entre l'application et le serveur de base de données.

```
connexion.Open();
```

Fermeture de la connexion

La fermeture d'une connexion est une étape très importante. En effet, une connexion non fermée est une connexion potentiellement perdue mais qui doit néanmoins être gérée par le serveur de base de données.

Cette fermeture doit être effectuée à l'aide de la méthode `Close` :

```
connexion.Close();
```

La classe `SqlConnection` implémente l'interface `IDisposable`. Il est donc pertinent d'utiliser la structure `using` (cf. chapitre Les bases du langage, section Autres structures) afin d'éviter tout blocage de la connexion en cas d'exception ou d'oubli de fermeture.

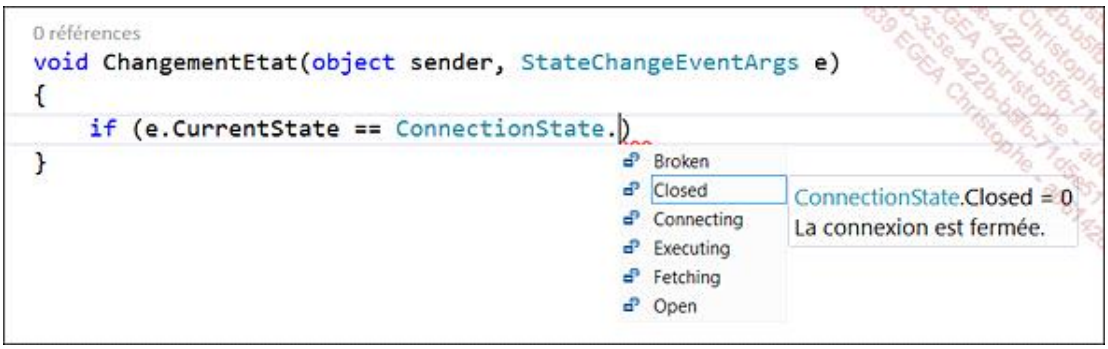
Événements

La classe `SqlConnection` possède deux événements permettant d'obtenir des informations sur la connexion.

L'événement `StateChanged` est déclenché lorsque l'état de la connexion est modifié, que cette modification soit initiée par l'utilisateur ou forcée par le serveur de base de données. Le gestionnaire pour cet événement a la signature suivante :

```
void ChangementEtat(object sender, StateChangeEventArgs e)
```

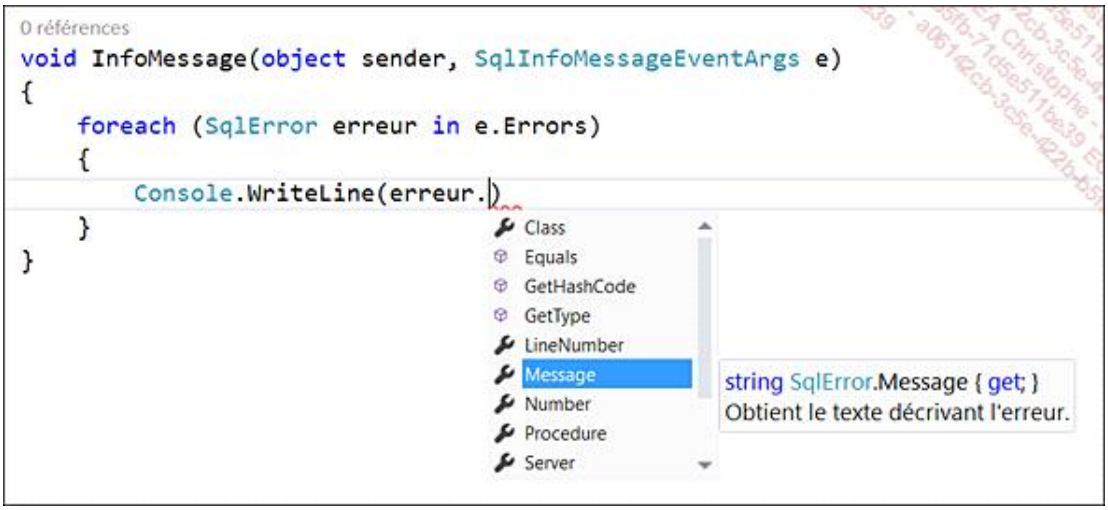
Le paramètre de type `StateChangeEventArgs` a deux propriétés : `OriginalState` et `CurrentState`. Elles permettent de connaître respectivement l'état original de la connexion et son nouvel état. Ces propriétés contiennent des valeurs de l'énumération `ConnectionState`, il est donc possible de tester facilement la valeur de ces éléments pour agir en conséquence.



L'événement `InfoMessage` permet quant à lui d'obtenir des informations concernant des situations anormales mais non critiques (sévérité inférieure à 10) qui se produisent sur la connexion. La signature du gestionnaire d'événements associé est la suivante :

```
void InfoMessage(object sender, SqlInfoMessageEventArgs e)
```

Le paramètre de type `SqlInfoMessageEventArgs` contient une collection d'objets `SqlErrors` correspondant à chacune des erreurs renvoyées par le serveur de base de données. Ces objets ont plusieurs propriétés permettant de diagnostiquer et de corriger le problème.



2. Création et exécution de commandes

Une fois la connexion au serveur de base de données établie, il est possible d'exécuter des commandes SQL au travers d'objets de type `SqlCommand`.

Ces commandes peuvent correspondre à tout type de requêtes SQL (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`, `ALTER TABLE`, etc.) mais elles peuvent aussi correspondre à une demande d'exécution d'une procédure stockée.

a. Définition et création d'une commande

La création d'un objet `SqlCommand` est effectuée de deux manières :

- En utilisant un des constructeurs de la classe `SqlCommand`. Ici, nous utiliserons le constructeur sans paramètre,

mais il est bien évidemment possible d'utiliser un des trois autres, sous réserve de pouvoir spécifier les paramètres nécessaires à leur utilisation.

```
SqlCommand commande = new SqlCommand();
```

Pour associer cette commande à une connexion, il faut affecter un objet `SqlConnection` à la propriété `Connection` de la commande.

```
commande.Connection = connexion;
```

- En utilisant la méthode `CreateCommand` de l'objet `SqlConnection`.

```
SqlCommand commande = connexion.CreateCommand();
```

Attention : pour pouvoir exécuter la commande, il est nécessaire que la connexion soit ouverte ! Assurez-vous de son état avant la création des commandes dont vous avez besoin, ceci vous évitera des déconvenues.

b. Sélection de données

La sélection de données est l'opération effectuée le plus fréquemment sur une base de données. Celle-ci est très simple à mettre en place à l'aide des objets `SqlConnection` et `SqlCommand`.

Après la création et l'association d'un objet `SqlCommand` à une connexion, il suffit de valoriser la propriété `CommandText` de la commande avec une requête SQL pour avoir une commande de sélection prête à être exécutée.

```
commande.CommandText = "SELECT * FROM Eleve";
```

L'exécution de requêtes de sélection de données est effectuée à l'aide de la méthode `ExecuteReader` de l'objet `SqlCommand`. Cette méthode renvoie un objet de type `SqlDataReader` grâce auquel il est possible de récupérer chacun des enregistrements du résultat de la requête, un par un, et uniquement en avançant dans la collection d'enregistrements.

```
SqlDataReader reader = commande.ExecuteReader();
```

La lecture d'un enregistrement se fait au travers de l'appel de la méthode `Read` de l'objet `SqlDataReader`. Cette méthode renvoie un booléen indiquant si un enregistrement a pu être lu. Si tel est le cas, alors l'objet `SqlDataReader` contient les données de l'enregistrement, qui sont alors manipulables à l'aide de l'opérateur `[]`.

```
reader.Read();
//Lecture de la valeur de la colonne "Nom" de l'enregistrement
string nom = (string)reader["Nom"];
```

Il est nécessaire de transtyper les valeurs récupérées de cette manière car elles sont exposées par l'objet

`SqlDataReader` sous la forme d'un `object`.

Il est également possible de récupérer la valeur souhaitée sous une forme fortement typée à l'aide d'une des méthodes spécialisées de la classe `SqlDataReader` : la méthode `GetString` pour obtenir une chaîne de caractères, `GetInt32` pour un entier, `GetDateTime` pour une date, etc. Ces méthodes prennent comme paramètre un entier représentant l'index de la colonne dans l'enregistrement.

```
reader.Read();
//Lecture de la valeur de la colonne placée en première position
//de l'enregistrement
string nom = reader.GetString(0)
```

La lecture de la totalité des données retournées est effectuée à l'aide d'une boucle `while` pour laquelle la condition entrante est la valeur retournée par l'appel de la méthode `Read`.

```
while (reader.Read())
{
    Console.WriteLine("Nom = {0}", reader.GetString(0));
}
```

Lorsqu'une requête ne renvoie qu'une valeur, la lecture du résultat peut être simplifiée en utilisant la méthode `ExecuteScalar` de la commande. Celle-ci renvoie une valeur de type `object` qu'il faut donc transtyper au besoin.

```
SqlCommand commande = connexion.CreateCommand();
commande.CommandText = "SELECT COUNT(*) FROM Eleve";

int nbEleves = (int)commande.ExecuteScalar();
Console.WriteLine("Il y a {0} élèves", nbEleves);
```

c. Actions sur les données

L'exécution d'une requête d'ajout, de suppression ou de mise à jour de données est effectuée d'une manière très similaire à la sélection de données. Une commande contenant une requête `UPDATE`, `INSERT` ou `DELETE` est créée et associée à une connexion.

```
SqlCommand commande = connexion.CreateCommand();
commande.CommandText = "INSERT INTO Eleve (Nom, Prenom) VALUES
('MARTIN', 'Michel')";
```

Une fois cette commande définie, l'appel de la méthode `ExecuteNonQuery` de l'objet `SqlCommand` envoie la requête au serveur de base de données. Sa valeur de retour est un entier représentant le nombre d'enregistrements affectés par la requête.

```
int nombreEnregistrementsAffectes = commande.ExecuteNonQuery();
```

Les commandes modifiant des éléments de structure de la base de données (ALTER TABLE, par exemple) sont également exécutées à l'aide de cette méthode.

d. Paramétrage d'une commande

La méthode la plus simple pour la construction de requêtes SQL est la **concaténation de chaînes de caractères**. Cette méthode a l'avantage d'être aisément compréhensible par tous, mais elle est délicate à mettre en place. En effet, elle mène souvent à des **erreurs de syntaxe** (oublis d'espaces ou de virgules, par exemple) et peut poser des **problèmes de sécurité** (injection SQL).

L'utilisation de **paramètres** permet de réduire le nombre de problèmes engendrés par la création d'une requête SQL :

- Une requête paramétrée peut être **définie sans concaténation**, ce qui réduit les risques d'erreurs de syntaxes.
- **Les valeurs de paramètres sont validées** au niveau de leur type et de leur longueur avant exécution de la requête. De plus, les chaînes de caractères sont échappées de manière à ne pas être interprétées comme des instructions SQL.

Ces deux points réduisent considérablement la surface d'attaque pour les injections SQL.

Ci-dessous un exemple de construction d'une commande SQL à partir de concaténation de chaînes :

```
string nom = "MARTIN";
string prenom = "Michel";
DateTime datenaissance = new DateTime(2004, 05, 28);

SqlCommand commande = connexion.CreateCommand();
commande.CommandText = "INSERT INTO Eleve (Nom, Prenom,
DateNaissance) VALUES ('" + nom + "', '" + prenom + "', '" +
datenaissance.ToString("dd/MM/yyyy") + "')";
```

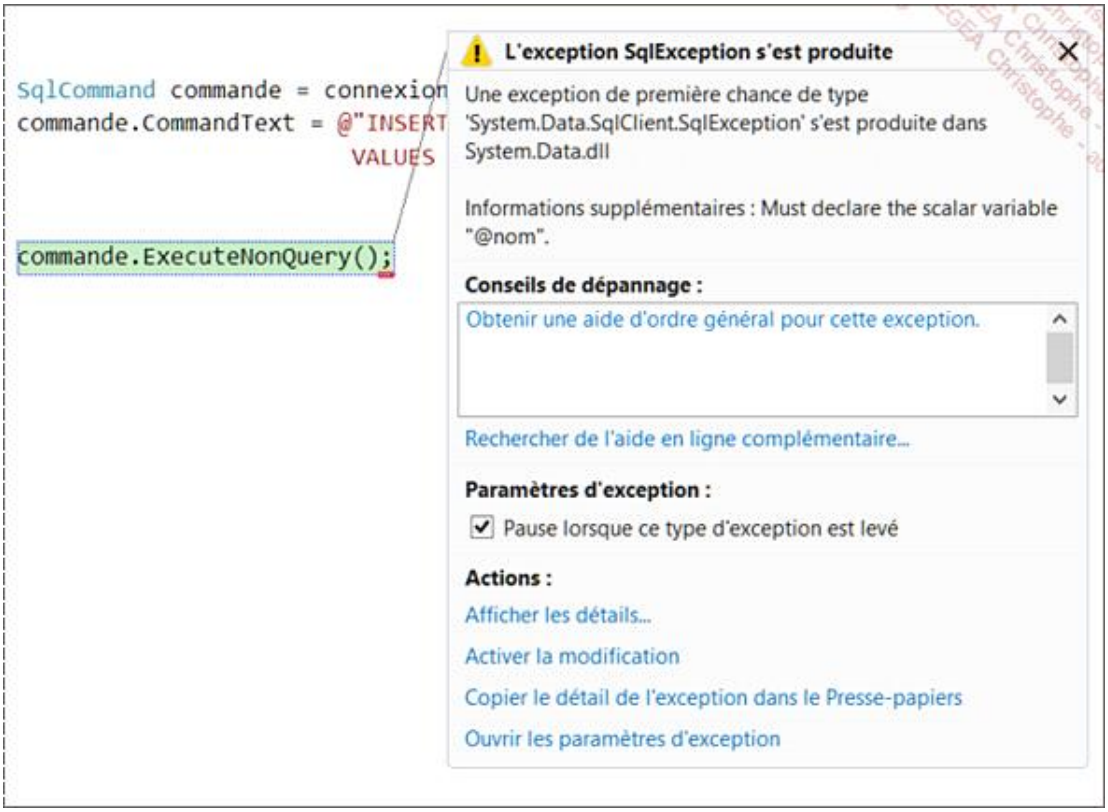
Les concaténations rendent la lecture difficile, et selon la culture utilisée par le serveur de base de données, il est possible d'obtenir un comportement inattendu (voire une exception) à cause du format de la date.

Il est possible de réécrire cette construction de commande en utilisant des paramètres de manière à rendre celle-ci plus maintenable et plus sécurisée.

Tout d'abord, il faut écrire la requête en définissant le nom et l'emplacement de ses paramètres : `@nom`, `@prenom`, `@datenaissance`.

```
SqlCommand commande = connexion.CreateCommand();
commande.CommandText = "INSERT INTO Eleve (Nom, Prenom,
DateNaissance) VALUES (@nom, @prenom, @datenaissance)";
```

Si la commande était exécutée en l'état, une exception serait levée car les valeurs de paramètres ne sont pas spécifiées.



La création de paramètres correspond à l'instanciation d'objets de type `SqlParameter` et à leur configuration. Il est notamment possible de spécifier le type attendu en base de données à l'aide de la propriété `DbType` ou la taille maximale du paramètre quand il s'agit d'une chaîne de caractères.

Le paramètre peut être défini comme étant en lecture seule (`ParameterDirection.Input`), en écriture seule (`ParameterDirection.Ouput`) ou en lecture et écriture (`ParameterDirection.InputOutput`) à l'aide de la propriété `Direction`. Ceci peut être très utile dans le cas de l'exécution d'une fonction ou procédure stockée.

```
string nom = "MARTIN";
string prenom = "Michel";
DateTime datenaissance = new DateTime(2004, 05, 28);

//Création du paramètre @nom
SqlParameter paramNom = new SqlParameter("@nom", nom);
paramNom.Direction = ParameterDirection.Input;
paramNom.DbType = DbType.String;
paramNom.Size = 30;

//Création du paramètre @prenom
SqlParameter paramPrenom = new SqlParameter("@prenom", prenom);
paramPrenom.Direction = ParameterDirection.Input;
paramPrenom.DbType = DbType.String;
paramPrenom.Size = 30;

//Création du paramètre @datenaissance
SqlParameter paramDate = new SqlParameter("@datenaissance",
datenaissance);
paramPrenom.Direction = ParameterDirection.Input;
paramPrenom.DbType = DbType.DateTime;
```

Pour terminer, il convient d'ajouter les paramètres créés à la collection de paramètres de la commande.

```
commande.Parameters.Add(paramNom);
commande.Parameters.Add(paramPrenom);
commande.Parameters.Add(paramDate);
```

e. Exécution de procédures stockées

Les procédures stockées sont des éléments comparables aux procédures C#, mais **exécutés entièrement par le serveur de base de données**. Elles sont composées d'instructions SQL (ou de leur variante spécifique au serveur de base de données) et peuvent accepter des paramètres, renvoyer des valeurs et effectuer des traitements complexes.

Leur utilisation a plusieurs avantages, notamment la **centralisation du code SQL**, ce qui peut simplifier la maintenance lorsqu'une base de données évolue sans impact sur le code des applications qui l'utilisent. De plus, les procédures stockées bénéficient d'un traitement particulier : elles sont **compilées** à leur première exécution par le moteur de base de données et mises en cache, ce qui peut apporter une amélioration conséquente de performances dans le cas de traitements complexes.

L'appel d'une procédure stockée à partir de C# est similaire à l'exécution de requêtes SQL. Après avoir créé un objet `SqlCommand`, il faut assigner à sa propriété `CommandText` le nom de la procédure stockée à exécuter et modifier la propriété `CommandType` avec la valeur `CommandType.StoredProcedure` pour indiquer que `CommandText` contient le nom de la procédure stockée.

Il convient ensuite de spécifier les paramètres attendus par la procédure stockée de la même manière que pour une requête SQL. Lorsqu'une procédure stockée renvoie une valeur (à l'aide de l'instruction `RETURN`), il convient d'ajouter un paramètre supplémentaire nommé `RETURN_VALUE` en assignant à sa propriété `Direction` la valeur `ParameterDirection.ReturnValue`. Après l'exécution de la procédure, il sera ainsi possible de récupérer la valeur de retour, comme pour les propriétés en écriture, au travers de la propriété `Value` du paramètre.

Pour tester ces différentes notions, nous allons utiliser une procédure stockée calculant l'âge moyen des élèves nés après une année donnée.

```
CREATE PROCEDURE CalculAgeMoyenElevesNeApresAnnee @annee int AS
    -- Déclaration de la variable qui contiendra l'âge moyen
    DECLARE @moyenne decimal

    -- Calcul de la moyenne
    SELECT @moyenne = AVG((Year(GetDate()) - Year(DateNaissance)))
          FROM Eleve
          WHERE YEAR(DateNaissance) > @annee

    --Renvoi de la variable @moyenne
    RETURN @moyenne
```

Le code C# permettant l'appel de cette procédure stockée est le suivant.

```
int annee = 2000;

SqlCommand commande = connexion.CreateCommand();
commande.CommandText = "CalculAgeMoyenElevesNeApresAnnee";
commande.CommandType = CommandType.StoredProcedure;

SqlParameter parametreAnnee = new SqlParameter("@annee", annee);
parametreAnnee.Direction = ParameterDirection.Input;
parametreAnnee.DbType = DbType.Int32;

SqlParameter parametreValeurRetour = new
SqlParameter("RETURN_VALUE", SqlDbType.Decimal);
parametreValeurRetour.Direction = ParameterDirection.ReturnValue;

commande.Parameters.Add(parametreAnnee);
commande.Parameters.Add(parametreValeurRetour);

commande.ExecuteNonQuery();

Console.WriteLine("L'âge moyen des élèves nés après {0} est {1}
ans", annee, parametreValeurRetour.Value);
```