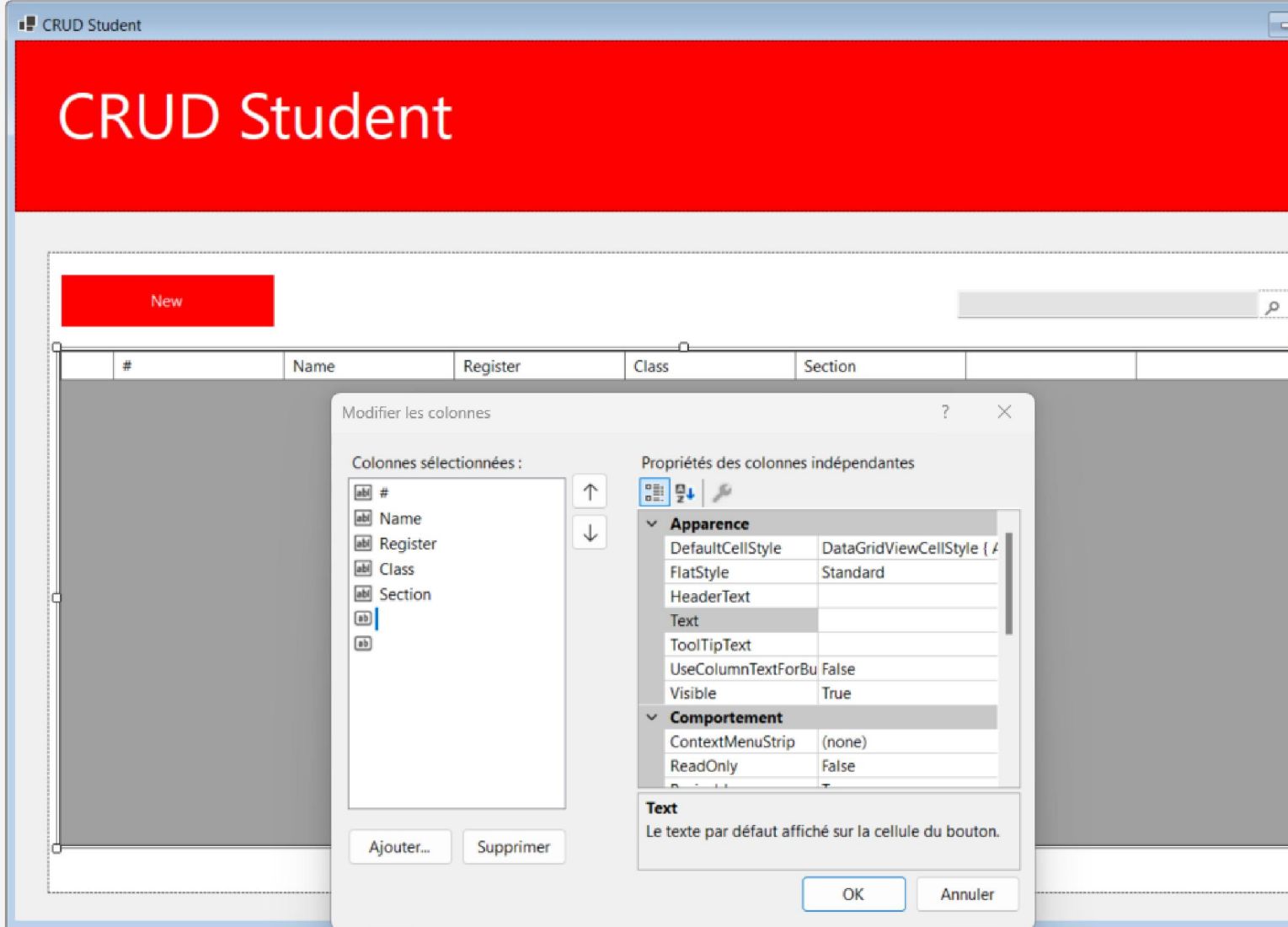




IGL245-Base de données et IHM- 2 crédits

1. Principes de création des interfaces Hommes/Machine
2. programmation événementielle
3. principales méthodes d'accès aux données (ADO, ODBC, OLE DB, ...)
4. TP en Visual Basic, .NET ou Développer



Homme/Machine

Définition

Les interfaces homme-machine (IHM) sont des systèmes qui permettent aux utilisateurs d'interagir avec des machines ou des logiciels. Une IHM efficace doit être intuitive, réactive et accessible.

Principes de conception

- **Clarté** : Les informations doivent être présentées de manière claire et compréhensible.
- **Consistance** : Utiliser des éléments d'interface cohérents pour éviter la confusion.
- **Feedback** : Fournir des retours immédiats sur les actions de l'utilisateur (ex. : confirmation d'une action).
- **Simplicité** : Éviter la surcharge d'informations et faciliter la navigation.
- **Accessibilité** : Assurer que l'interface est utilisable par tous, y compris les personnes handicapées.

Méthodes de conception

- **Prototypage** : Créer des maquettes pour tester les idées avant la mise en œuvre.
- **Tests utilisateurs** : Évaluer l'interface avec de vrais utilisateurs pour identifier les problèmes et améliorer l'expérience.

Exemples

- En JavaScript, l'ajout d'un gestionnaire d'événements à un bouton :

```
document.getElementById("monBouton").addEventListener("click", function() {  
    alert("Bouton cliqué !");  
});
```

1. Principes de création des interfaces Hommes/Machine (IHM) avec .NET

1.1 Ergonomie et conception

- **Objectif de l'IHM** :
 - Créez une interface intuitive, agréable et accessible.
 - Respecter les bonnes pratiques ergonomiques (disposition cohérente, lisibilité, contraste, etc.).
- **Conception centrale sur l'utilisateur** :
 - Recueillir les besoins via des questionnaires, tests utilisateurs, et itérer à partir de prototypes.
- **Outils .NET** :
 - **Windows Forms** ou **WPF (Windows Presentation Foundation)** offrent des environnements de conception visuelle intégrés dans Visual Studio.
 - Le Designer intégré permet de positionner, redimensionner et paramétrer les contrôles graphiques (boutons, zones de texte, listes, etc.).

1.2 Architecture et modèles de conception

- **Modèle MVC/MVVM** :
 - **MVC (Modèle-Vue-Contrôleur)** : Sépare la logique de l'application, l'interface et le contrôle des actions.
 - **MVVM (Model-View-ViewModel)** : Très utilisé en WPF, il facilite la liaison de données entre la vue et la logique métier.
- **Séparation des préoccupations** :

- L'IHM (la Vue) ne doit pas contenir de logique d'accès aux données. Celle-ci sera déléguée à la couche métier ou au modèle.
-

2. Programmation événementielle en .NET

2.1 Concepts de base

- **Qu'est-ce qu'un événement ?**
 - Un événement représente une action (clic, saisie, changement de sélection, etc.) déclenchée par l'utilisateur.
- **La boucle des événements :**
 - Le framework .NET gère une boucle qui attend et distribue les événements aux contrôles concernés.

2.2 Gestion des Événements et Délégués

- **Gestionnaires d'événements :**
 - En .NET, chaque contrôle expose des événements (par exemple, le bouton possède un événement Click).
 - On associe un gestionnaire à un événement soit via le Designer (double-clic sur le contrôle), soit par code.
- **Exemple en Visual Basic .NET (Windows Forms) :**

```
Public Class Form1
    ' Cette méthode est déclenchée lors du clic sur btnLoadData
    Private Sub btnLoadData_Click(sender As Object, e As EventArgs) Handles btnLoadData.Click
        MessageBox.Show("Bouton cliqué !")
    End Sub
End Class
```

- Ici, le mot-clé `Handles btnLoadData.Click` indique que la méthode gère l'événement Click du bouton.

2.3 Propagation et priorisation

- **Propagation des événements :**
 - Certains frameworks (notamment WPF) permettent de « bubler » ou « capturer » les événements, afin de traiter des actions à différents niveaux de la hiérarchie visuelle.
 - **Avantages de la programmation événementielle :**
 - Réactivité et modularité : chaque composant réagit à ses propres événements, simplifiant ainsi le débogage et la maintenance.
-

3. Principales méthodes d'accès aux données dans .NET

3.1 La couche d'accès aux données

- **Objectif :**
 - Fournir une interface unifiée pour interagir avec des bases de données (SQL Server, Oracle, etc.) ou d'autres sources de données.

- **Séparation logique :**
 - L'accès aux données est déconnecté de la logique métier et de l'IHM, facilitant ainsi la maintenance et la réutilisation.

3.2 ADO.NET

- **Présentation :**
 - ADO.NET est l'API d'accès aux données pour le framework .NET. Elle permet de se connecter, d'exécuter des commandes SQL et de manipuler des ensembles de données.
- **Objets clés :**
 - **SqlConnection** : Gère la connexion à la base de données.
 - **SqlCommand** : Permet d'exécuter des commandes SQL (SELECT, INSERT, UPDATE, DELETE).
 - **SqlDataAdapter** : Facilite le remplissage d'un DataSet ou DataTable avec les résultats d'une requête.
 - **DataSet/DataTable** : Structures en mémoire contenant les données récupérées.

3.3 ODBC et OLE DB (et Autres)

- **ODBC :**
 - Interface standard pour diverses bases de données. En .NET, on peut utiliser la classe OdbcConnection pour se connecter via un pilote ODBC.
- **Base de données OLE :**
 - Une autre technologie Microsoft qui permet d'accéder à des sources de données hétérogènes. Moins utilisé aujourd'hui en faveur d'ADO.NET et d'ODBC.

3.4 Exemple d'accès aux données avec ADO.NET (VB.NET)

Imports System.Data.SqlClient

Public Class Form1

' Gestionnaire de l'événement Click du bouton de chargement

Private Sub btnLoadData_Click(sender As Object, e As EventArgs) Handles btnLoadData.Click

' Chaîne de connexion à la base de données SQL Server

Dim connectionString As String = "Data Source=.;Initial Catalog=MaBase;Integrated Security=True"

Using conn As New SqlConnection(connectionString)

Try

conn.Open()

Dim query As String = "SELECT * FROM Clients"

Dim cmd As New SqlCommand(query, conn)

Dim adapter As New SqlDataAdapter(cmd)

Dim dt As New DataTable()

adapter.Fill(dt)

' Liaison des données récupérées à un DataGridView pour affichage

DataGridView1.DataSource = dt

Catch ex As Exception

MessageBox.Show("Erreur de connexion : " & ex.Message)

End Try

End Using

End Sub

End Class

- Cet exemple montre comment ouvrir une connexion à une base de données, exécuter une requête SQL et afficher les résultats dans un contrôle de type DataGridView.

4. TP Pratique sur .NET

4.1 Objectifs du TP

- **Concevoir et développer** une application Windows Forms en VB.NET (ou C#) qui combine :
 - Une interface graphique ergonomique.
 - La programmation événementielle pour réagir aux actions de l'utilisateur.
 - L'accès aux données via ADO.NET pour interagir avec une base de données SQL Server.

4.2 Étapes du TP

4.2.1 Création de l'interface utilisateur

- **Utilisez le Designer Windows Forms** dans Visual Studio pour créer une fenêtre principale.
- **Ajouter des contrôles** tels que :
 - Un **DataGridView** pour afficher les données.
 - Des **boutons** pour charger, enregistrer ou mettre à jour les données.
 - Des **TextBox** et **Label** pour saisir des informations.

4.2.2 Mise en œuvre de la programmation événementielle

- **Associer des événements** aux contrôles (par exemple, le clic sur un bouton déclenche la récupération des données).
- **Exemple :**

```
' Dans le Designer, double-cliquez sur le bouton "Charger" pour générer le gestionnaire d'événement.  
Private Sub btnCharger_Click(sender As Object, e As EventArgs) Handles btnCharger.Click  
    ChargerDonnees()  
End Sub
```

```
Private Sub ChargerDonnees()  
    ' Code d'accès aux données (similaire à l'exemple ADO.NET précédent)  
End Sub
```

4.2.3 Accès aux données et liaison de données

- **Utilisez ADO.NET** pour se connecter à la base de données et exécuter des requêtes.
- **Remplissez un DataTable ou DataSet** et liez ce dernier au DataGridView afin que les données s'affichent automatiquement.
- **Implémenter des fonctionnalités** d'ajout, de modification et de suppression de données via des commandes SQL exécutées à partir de la couche d'accès.

4.2.4 Test et débogage

- **Tester l'application** en exécutant plusieurs scénarios (chargement des données, modification, enregistrement).
- **Utiliser les outils de Visual Studio** (débogueur, points d'arrêt) pour vérifier le flux des événements et la manipulation des données.

4.3 Conclusion du TP

- **Synthèse du TP :**
 - Vous aurez créé une application complète qui intègre une IHM réactive et une connexion sécurisée à une base de données.
 - Vous aurez mis en œuvre la programmation événementielle pour gérer les interactions utilisateur.
 - Vous utiliserez ADO.NET pour réaliser l'accès et la manipulation des données.
 - **Perspectives :**
 - Évolution vers des architectures plus avancées (par exemple, en passant à WPF et le pattern MVVM) et l'intégration de services web pour un accès aux données plus distribuées.
-

Conclusion

Ce cours sur le framework .NET vous a permis de :

1. Comprendre les **principes de création des interfaces Hommes/Machine** en s'appuyant sur Windows Forms ou WPF, avec une attention particulière à l'ergonomie et à la conception centrale entrée utilisateur.
2. Maîtriser la **programmation événementielle** en .NET, avec des exemples de gestion d'événements et l'utilisation de délégués.
3. Découvrez les **principales méthodes d'accès aux données** en .NET, en mettant l'accent sur ADO.NET ainsi que sur les autres technologies comme ODBC et OLE DB.
4. Réaliser un **TP pratique** dans lequel vous concevez une application complète combinant IHM et accès aux données, illustrée ici avec des exemples en Visual Basic .NET.

Ce module vous offre ainsi les bases pour développer des applications riches et interactives en environnement .NET. N'hésitez pas à approfondir chaque point en consultant la documentation officielle de Microsoft et à expérimenter avec vos propres projets.

```





namespace crud_01
{
    partial class FormStudentInfo
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        private void InitializeComponent()
        {
            SuspendLayout();

            AutoScaleDimensions = new.SizeF(8F, 20F);
            AutoScaleMode = AutoScaleMode.Font;
            ClientSize = new.Size(630, 520);
            Name = "FormStudentInfo";
            Text = "Form1";
            ResumeLayout(false);
        }
    }
}

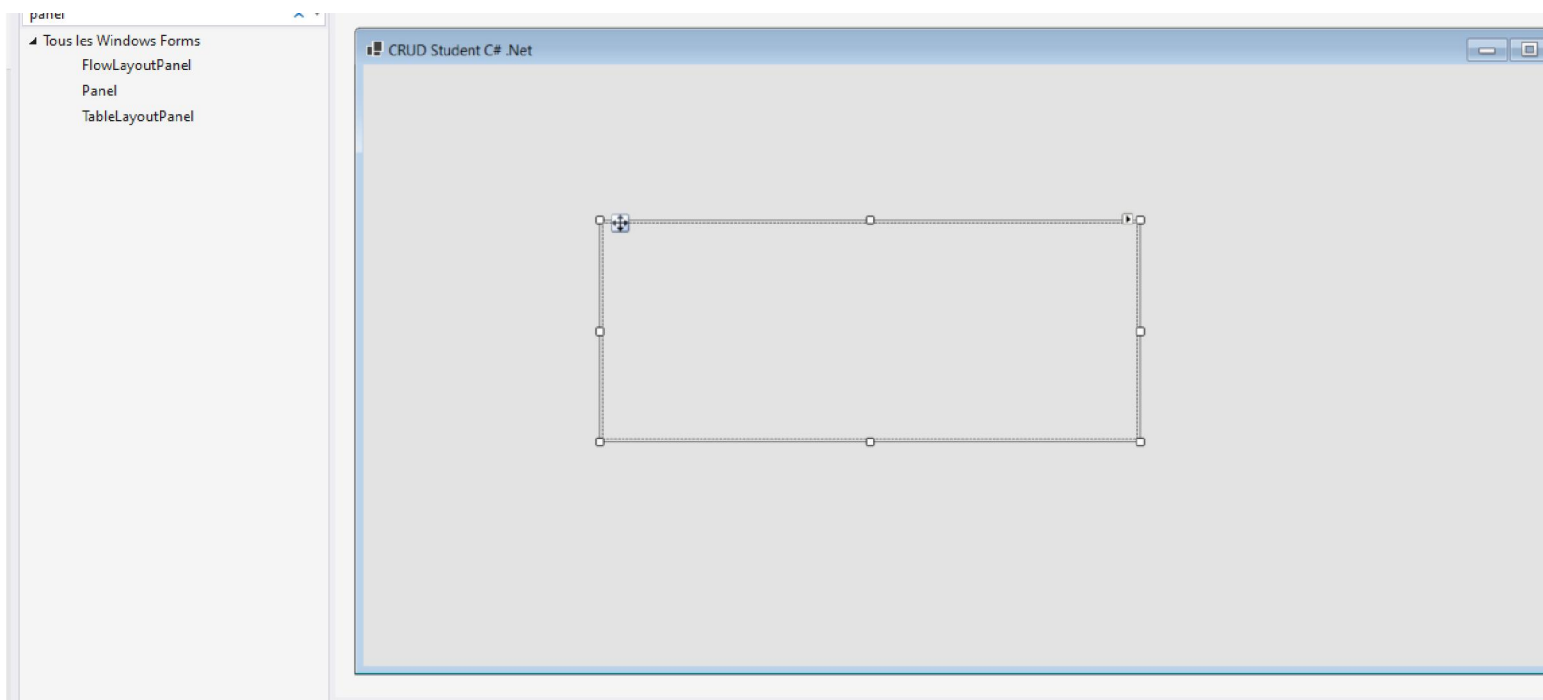
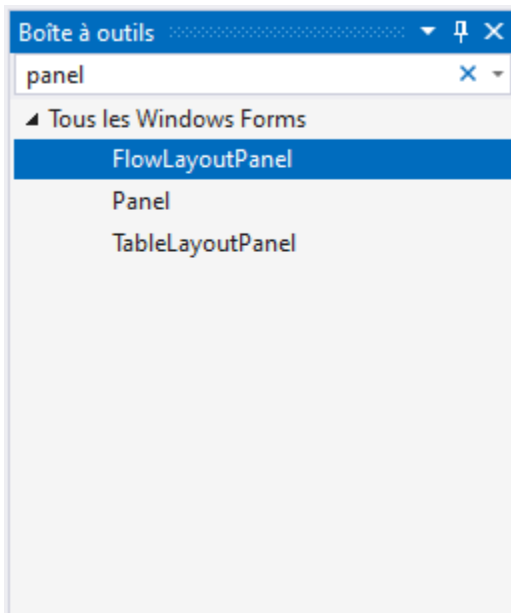
```

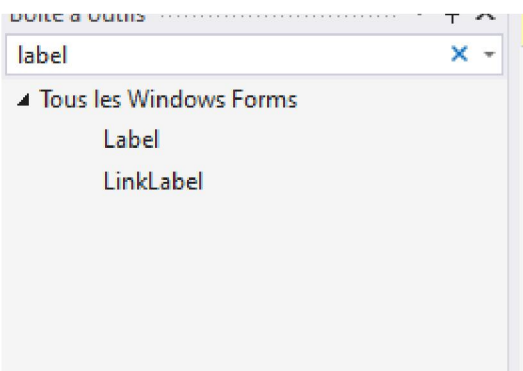
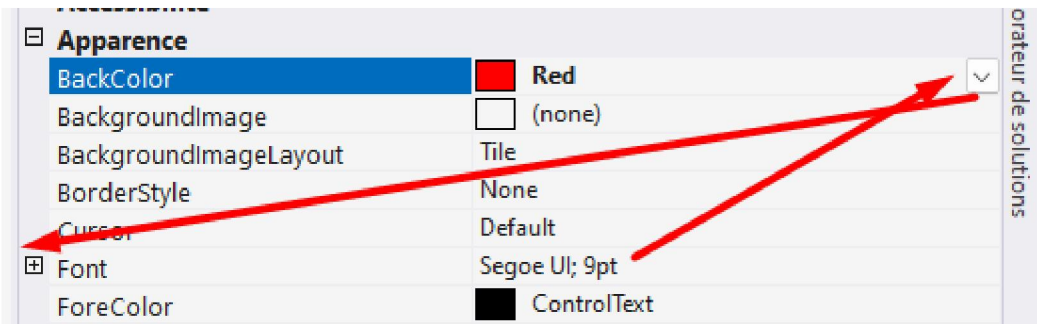
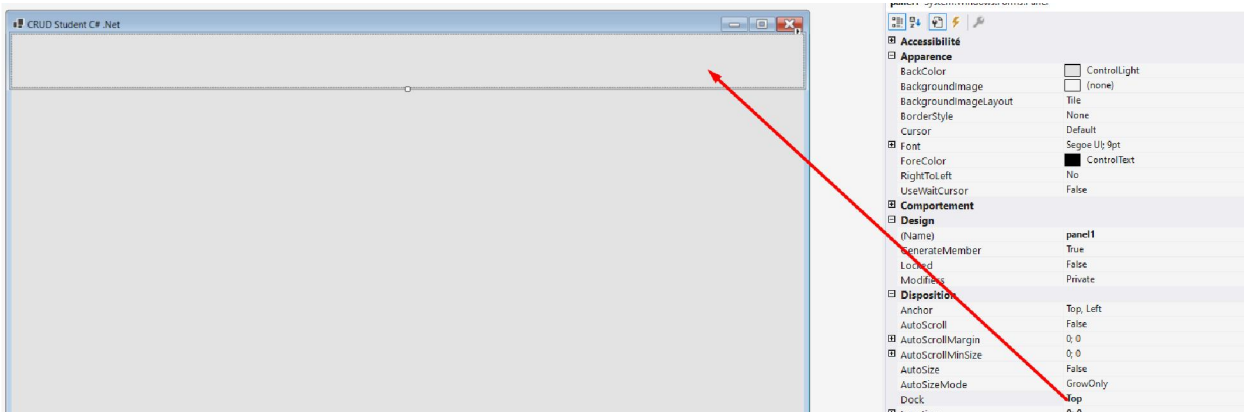
[-] Apparence	
BackColor	 ControlLight
BackgroundImage	 (none)
BackgroundImageLayout	Tile
Cursor	Default
[-] Font	
Name	 Segoe UI
Size	9
Unit	Point
Bold	False
GdiCharSet	1
GdiVerticalFont	False
Italic	False
Strikeout	False
Underline	False
ForeColor	 ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1
UseWaitCursor	False

[-] Comportement	
AllowDrop	False
AutoValidate	EnablePreventFocusChange
ContextMenuStrip	(none)
DoubleBuffered	False
Enabled	True
ImeMode	NoControl
[-] Design	
(Name)	Form1
Language	(Default)
Localizable	False
Locked	False

⊕	Accessibilité	
⊖	Apparence	
	BackColor	<input type="text"/> ControllLight
	BackgroundImage	<input type="text"/> (none)
	BackgroundImageLayout	Tile
	Cursor	Default
⊕	Font	Segoe UI; 9pt
	ForeColor	<input type="text"/> ControlText
	FormBorderStyle	Sizable
	RightToLeft	No
	RightToLeftLayout	False
	Text	CRUD Student C# .Net
	UseWaitCursor	False
⊕	Comportement	
⊖	Design	
	(Name)	FormStudent
	Language	(Default)
	Localizable	False
	Locked	False
⊖	Disposition	
	AutoScaleMode	Font
	AutoScroll	False
⊕	AutoScrollMargin	0; 0
⊕	AutoScrollMinSize	0; 0
	AutoSize	False
	AutoSizeMode	GrowOnly
⊕	Location	0; 0
⊕	MaximumSize	0; 0
⊕	MinimumSize	0; 0
⊕	Padding	0; 0; 0; 0
⊕	Size	1281; 663
	StartPosition	CenterScreen
	WindowState	Normal
⊖	Données	
⊕	(DataBindings)	(ControlBindings)
	Tag	
⊖	Focus	
	CausesValidation	True
⊖	Misc	
	AcceptButton	(none)

⊕	Comportement	
⊖	Design	
	(Name)	Form1
	Language	(Default)
	Localizable	False
	Locked	False
⊖	Disposition	
	AutoScaleMode	Font
	AutoScroll	False
⊖	AutoScrollMargin	0; 0
	Width	0
	Height	0
⊖	AutoScrollMinSize	0; 0
	Width	0
	Height	0
	AutoSize	False
	AutoSizeMode	GrowOnly
⊖	Location	0; 0
	X	0
	Y	0
⊖	MaximumSize	0; 0
	Width	0
	Height	0
⊖	MinimumSize	0; 0
	Width	0
	Height	0
⊖	Padding	0; 0; 0; 0
	All	0
	Left	0
	Top	0
	Right	0
	Bottom	0
⊖	Size	1281; 663
	Width	1281
	Height	663
	StartPosition	WindowsDefaultLocation
	WindowState	Normal
⊖	Données	
⊕	(DataBindings)	(ControlBindings)
	Tag	





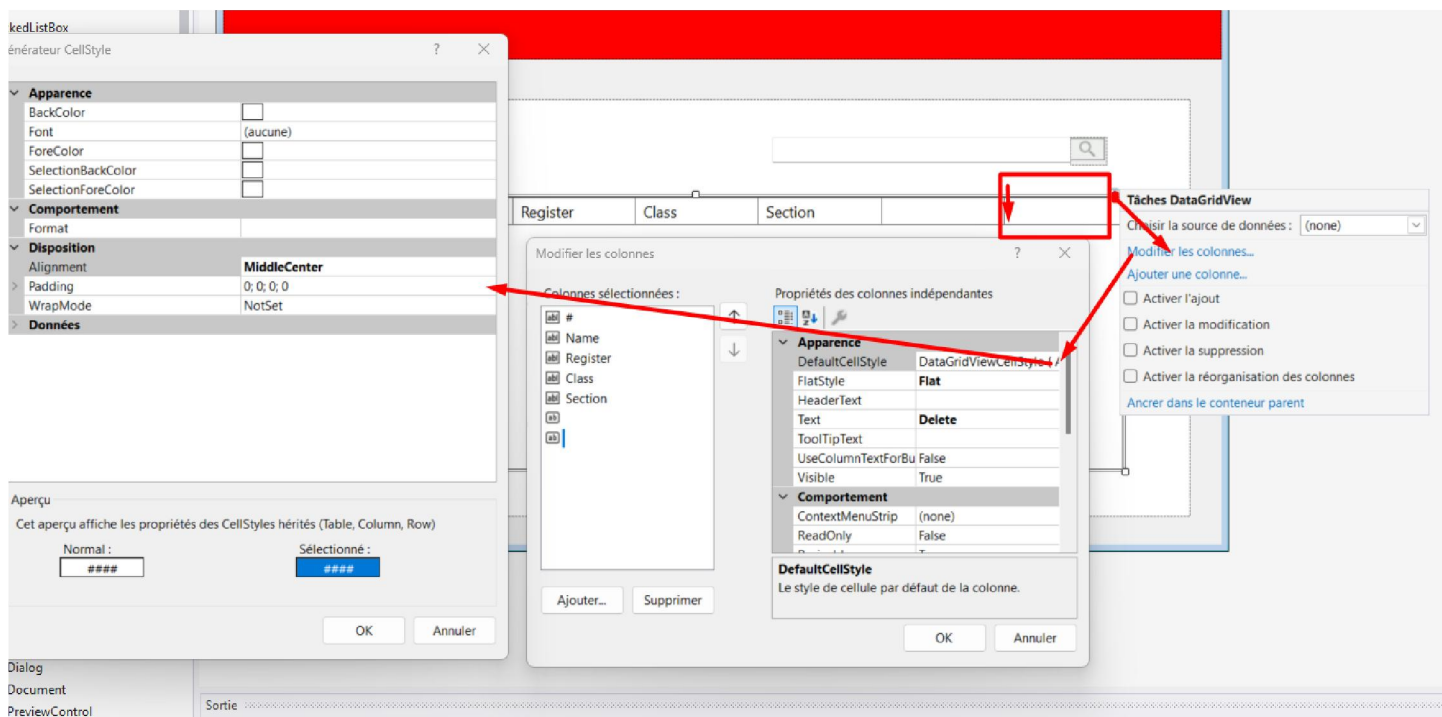
Font	Segoe UI; 9pt
Name	ab Segoe UI
Size	9
Unit	Point
Bold	False
GdiCharSet	1
GdiVerticalFont	False
Italic	False
Strikeout	False
Underline	False
ForeColor	ControlText
RightToLeft	No
UseWaitCursor	False
Comportement	
AllowDrop	False
ContextMenuStrip	(none)
Enabled	True
ImeMode	NoControl
TabIndex	0
TabStop	False
Visible	True
Design	
Disposition	
Données	
(DataBindings)	(ControlBindings)
Tag	
Focus	











Enabled	True
ImeMode	NoControl
TabIndex	0
TabStop	False
Visible	True
Design	
(Name)	panel2
GenerateMember	True
Locked	False
Modifiers	Private
Disposition	
Anchor	Top, Bottom, Left, Right
AutoScroll	False
AutoScrollMargin	0; 0
AutoScrollMinSize	0; 0
AutoSize	False
AutoSizeMode	GrowOnly
Dock	None
Location	100; 273
Margin	3; 3; 3; 3
MaximumSize	0; 0
MinimumSize	0; 0
Padding	0; 0; 0; 0
Size	1053; 480
Width	1053
Height	480
Données	


New

#	Name	Register	Class	Section

ReadOnly	true
RowHeadersWidthSizeMode	EnableResizing
SelectionMode	RowHeaderSelect
StandardTab	False
TabIndex	3
TabStop	True
VirtualMode	False
Visible	True
Design	
(Name)	dataGridView1
GenerateMember	True
Locked	False
Modifiers	Private
Disposition	
Anchor	Top, Bottom, Left, Right
AutoSizeColumnsMode	Fill
AutoSizeRowsMode	None
Dock	None
Location	11; 93
Margin	3; 3; 3; 3
MaximumSize	0; 0
MinimumSize	0; 0
RowHeadersWidth	51
ScrollBars	Both
Size	1187; 471
Données	
(DataBindings)	(ControlBindings)
DataMember	
DataSource	(none)



EnableHeadersVisualStyles	True
GridColor	 WindowFrame
RightToLeft	No
RowHeadersBorderStyle	Raised
RowHeadersDefaultCellStyle	DataGridViewCellStyle { BackColor= Color [Control],
 RowHeadersVisible	False
RowsDefaultCellStyle	DataGridViewCellStyle { }
 RowTemplate	DataGridViewRow { Index=-1 }
ShowCellErrors	True
ShowCellToolTips	True
 ShowEditingIcon	False
ShowRowErrors	True
UseWaitCursor	False
 Comportement	
AllowDrop	False
AllowUserToAddRows	False
AllowUserToDeleteRows	False
AllowUserToOrderColumns	False
 AllowUserToResizeColumns	False
 AllowUserToResizeRows	False
ClipboardCopyMode	EnableWithAutoHeaderText
ColumnHeadersHeightSizeMode	AutoSize
ContextMenuStrip	(none)
EditMode	EditOnKeystrokeOrF2
Enabled	True
ImeMode	NoControl
 MultiSelect	False
ReadOnly	True
 RowHeadersWidthSizeMode	DisableResizing
SelectionMode	RowHeaderSelect
StandardTab	False
TabIndex	0
TabStop	True
VirtualMode	False
Visible	True
 Design	

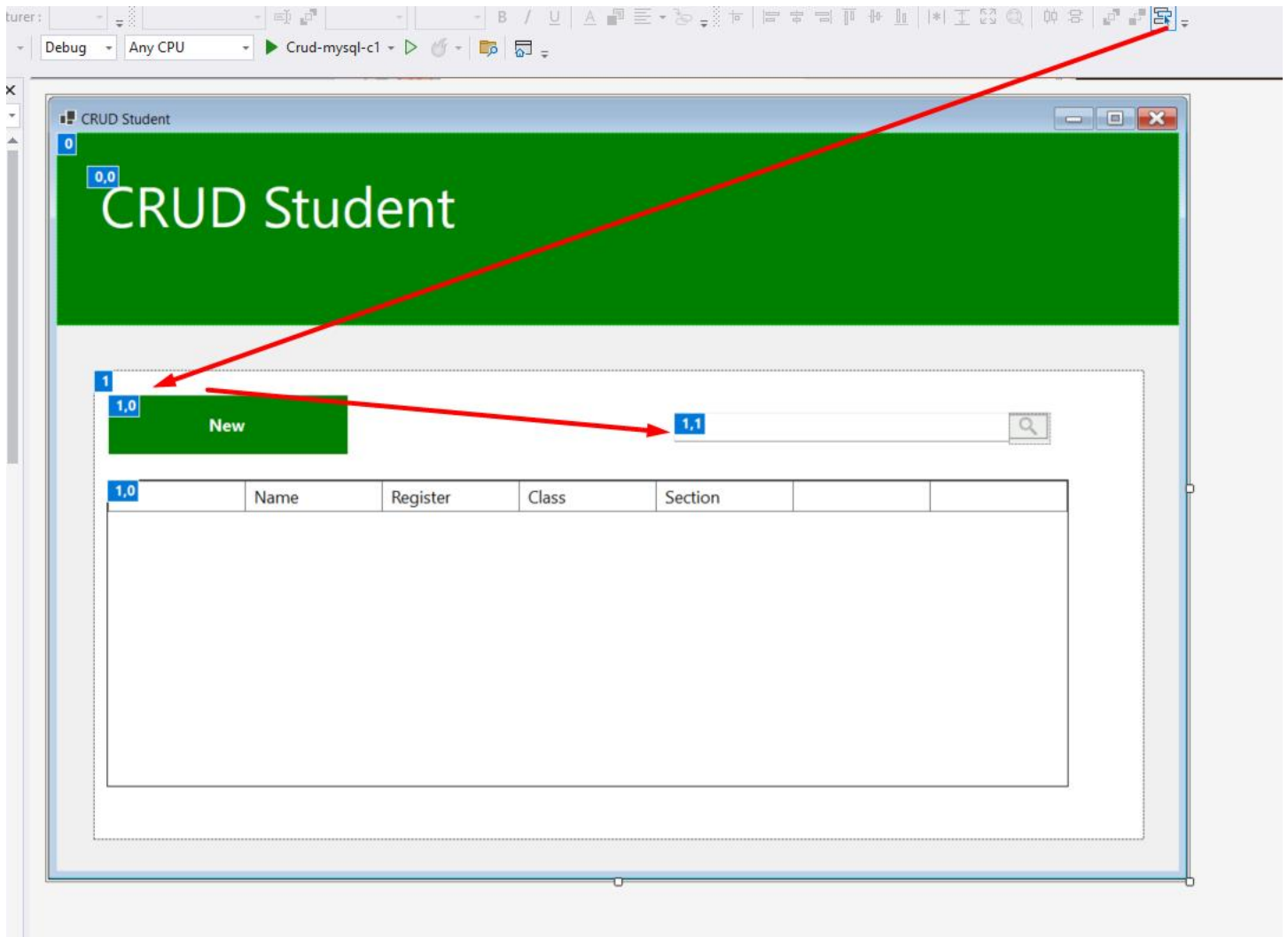
Visible	True
 Design	
(Name)	dataGridView
GenerateMember	True
Locked	False
Modifiers	Private

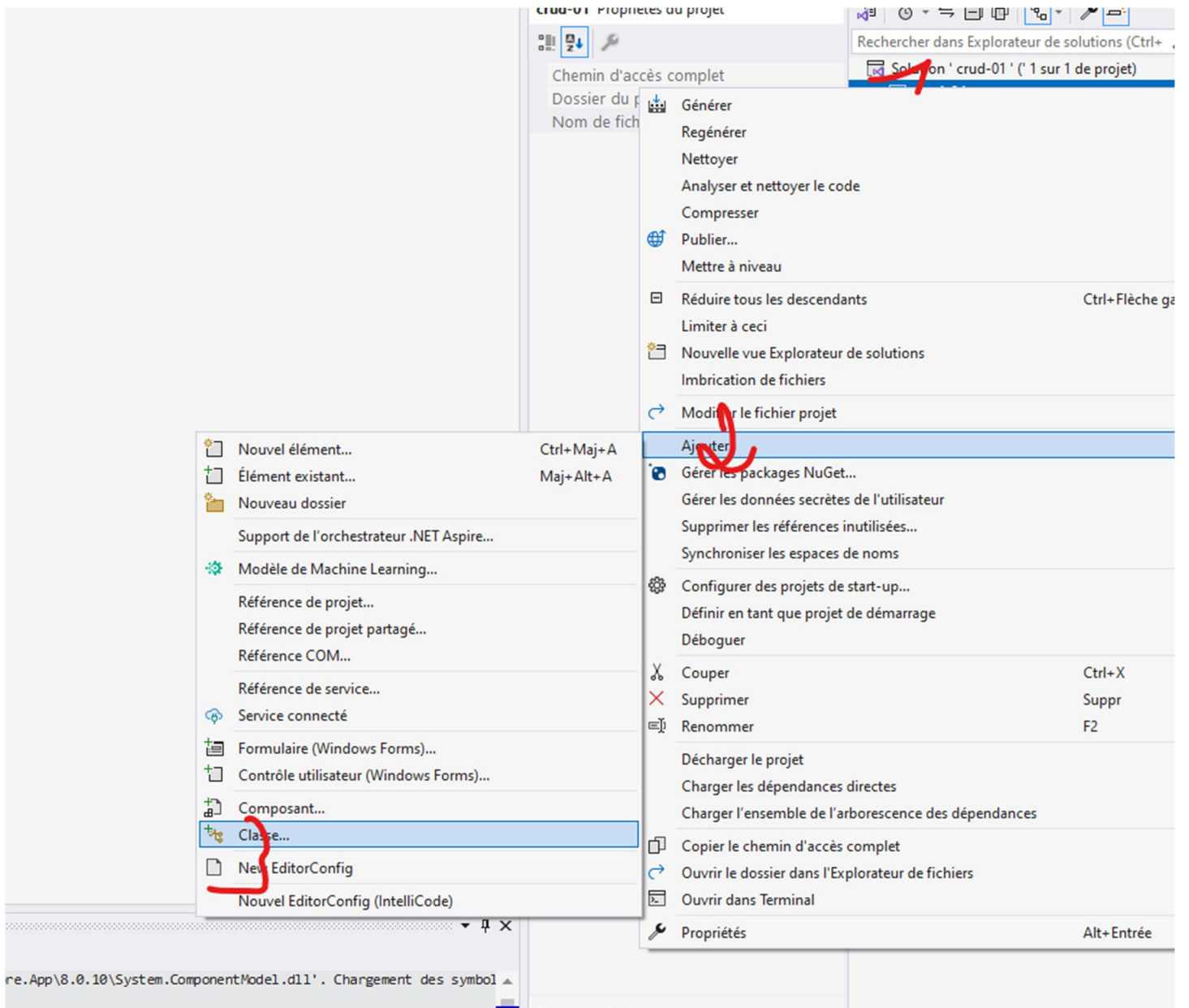
CRUD Student

New



#	Name	Register	Class	Section		





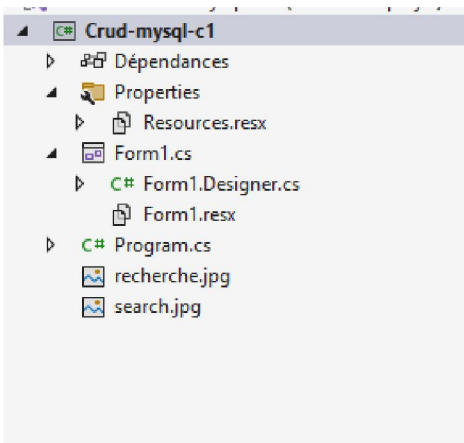
CRUD Student

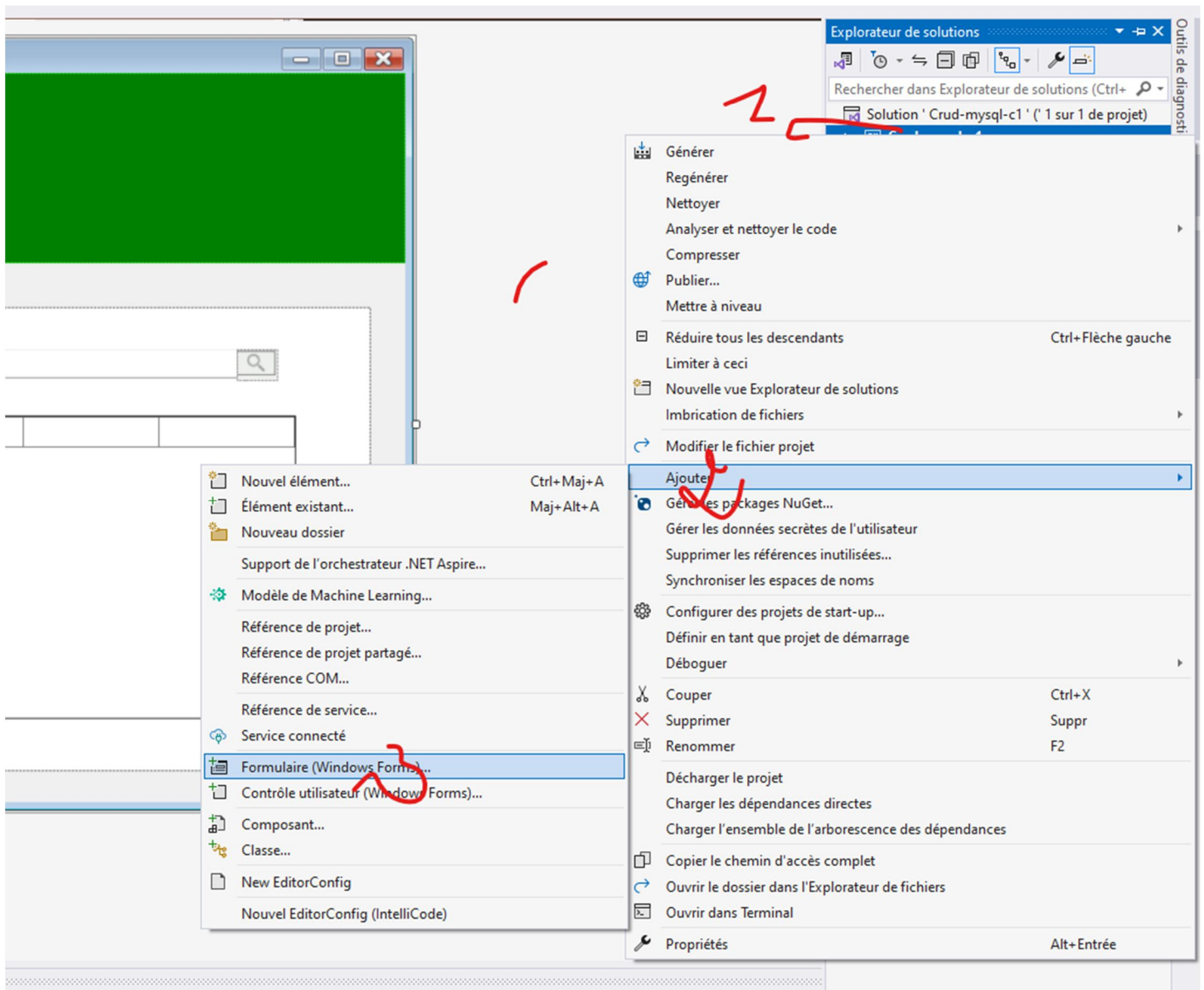
New

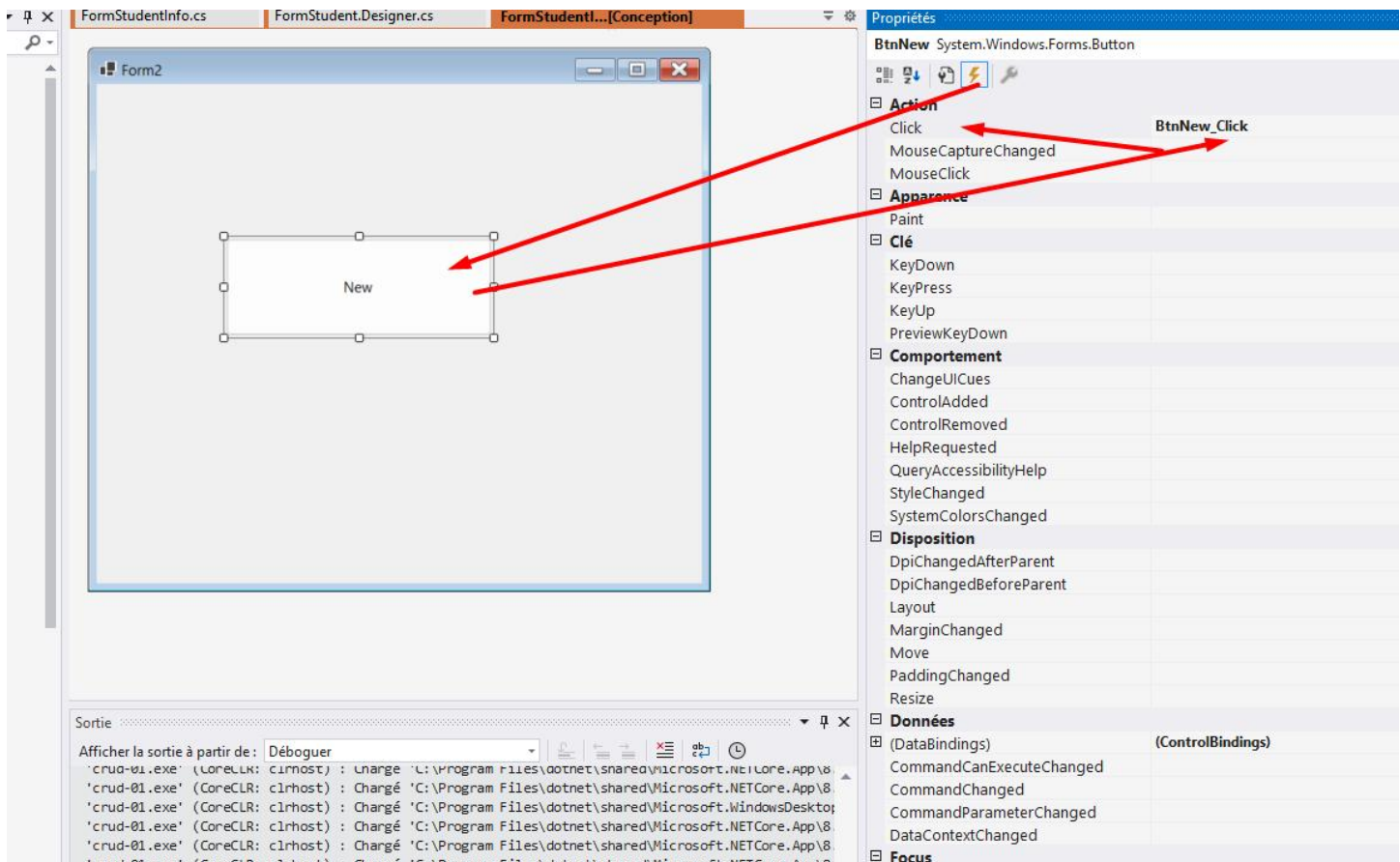
0.2

0.0	Name	Register	Class	Section

Crud-mysql-c1







Il semble que vous ayez deux formulaires dans votre application Windows Forms : `FormStudentInfo` et `FormStudentok`. Vous souhaitez ouvrir `FormStudentok` en cliquant sur un bouton dans `FormStudentInfo`, mais cela ne fonctionne pas actuellement. Voici quelques étapes pour résoudre ce problème :

1. Vérifiez que l'événement `click` du bouton est correctement associé à la méthode `BtnNew_Click` :

Assurez-vous que le nom de la méthode dans le code correspond exactement au nom de l'événement associé au bouton dans le concepteur de formulaires. Pour ce faire, sélectionnez le bouton dans le concepteur, allez dans la fenêtre des propriétés, cliquez sur l'icône des événements (en forme d'éclair) et vérifiez que l'événement `Click` est bien associé à `BtnNew_Click`.

2. Ajoutez du code à la méthode `InitializeComponent` de `FormStudentok` :

Si la méthode `InitializeComponent` est vide ou absente, le formulaire ne sera pas correctement initialisé. Cette méthode est généralement générée automatiquement par le concepteur de Visual Studio et contient le code nécessaire pour initialiser les contrôles du formulaire. Assurez-vous que cette méthode est présente et correctement implémentée dans `FormStudentok`.

3. Vérifiez les exceptions ou erreurs à l'exécution :

Il est possible qu'une exception soit levée lors de l'exécution, empêchant l'ouverture du formulaire. Pour diagnostiquer cela, vous pouvez entourer le code d'ouverture du formulaire avec un bloc `try-catch` et afficher toute exception :

```
private void BtnNew_Click(object sender, EventArgs e)
{
    try
    {
        FormStudentok form = new FormStudentok();
        form.ShowDialog();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Une erreur est survenue : " + ex.Message);
    }
}
```

Cela vous permettra de voir si une erreur spécifique empêche l'ouverture du formulaire.



Name

Register

Class

Section

Save

Nouvel onglet x Ouvrir formulaire ne fonctionne x rosto-infinity (ROSTAND LELE) x

localhost/phpmyadmin/index.php?route=/server/databases

phpMyAdmin

Récentes Préférées

- Nouvelle base de données
- ado-net-poo-student
- ado-net01
- bureau_etude
- carisma-dev
- cfpcanadienne
- cours-wp
- crud-php-2024-2025
- crud01-24-25
- crud02-24-25
- electromarket

Serveur : 127.0.0.1

Bases de données SQL État Comptes utilisateurs Exp

Bases de données

Création d'une base de données

student-dot-net utf8mb4_general_ci **Créer**

☐ Tout cocher **Supprimer**

Base de données	Interclassement	Action
<input type="checkbox"/> ado-net-poo-student	utf8mb4_general_ci	Vérifier les privilèges

Structure SQL Rechercher Requête Exporter Importer Opérations Privilèges Procédures stockées Évènements

Nom de table: student Ajouter 1 colonne(s) Exécuter

Nom	Type	Taille/Valeurs*	Valeur par défaut	Interclassement	Attributs	Null	Index	A_I	Comm
ID	INT		Aucun(e)			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	
Name	VARCHAR	100	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
Register	VARCHAR	100	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
Class	VARCHAR	100	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
Section	VARCHAR	100	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
Creat_at	TIMESTAMP		CURRENT_TIME			<input type="checkbox"/>	---	<input type="checkbox"/>	

Commentaires de table :

Interclassement :

Moteur de stockage : InnoDB

Parcourir

Structure

SQL

Rechercher

Insérer

Exporter

Importer

Privileges

Operacions

Déclencheurs

Structure de table

Vue relationnelle

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 ID	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
<input type="checkbox"/>	2 Name	varchar(100)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	3 Register	varchar(100)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	4 Class	varchar(100)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	5 Section	varchar(100)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	6 Creat_at	timestamp			Non	current_timestamp()			Modifier Supprimer Plus

☐ Tout cocher
 Avec la sélection :

Parcourir

Modifier

Supprimer

Primaire

Unique

Index

Spatial

Texte entier

Imprimer

Suggérer des optimisations de structure

Déplacer des colonnes

Normaliser

Ajouter

1

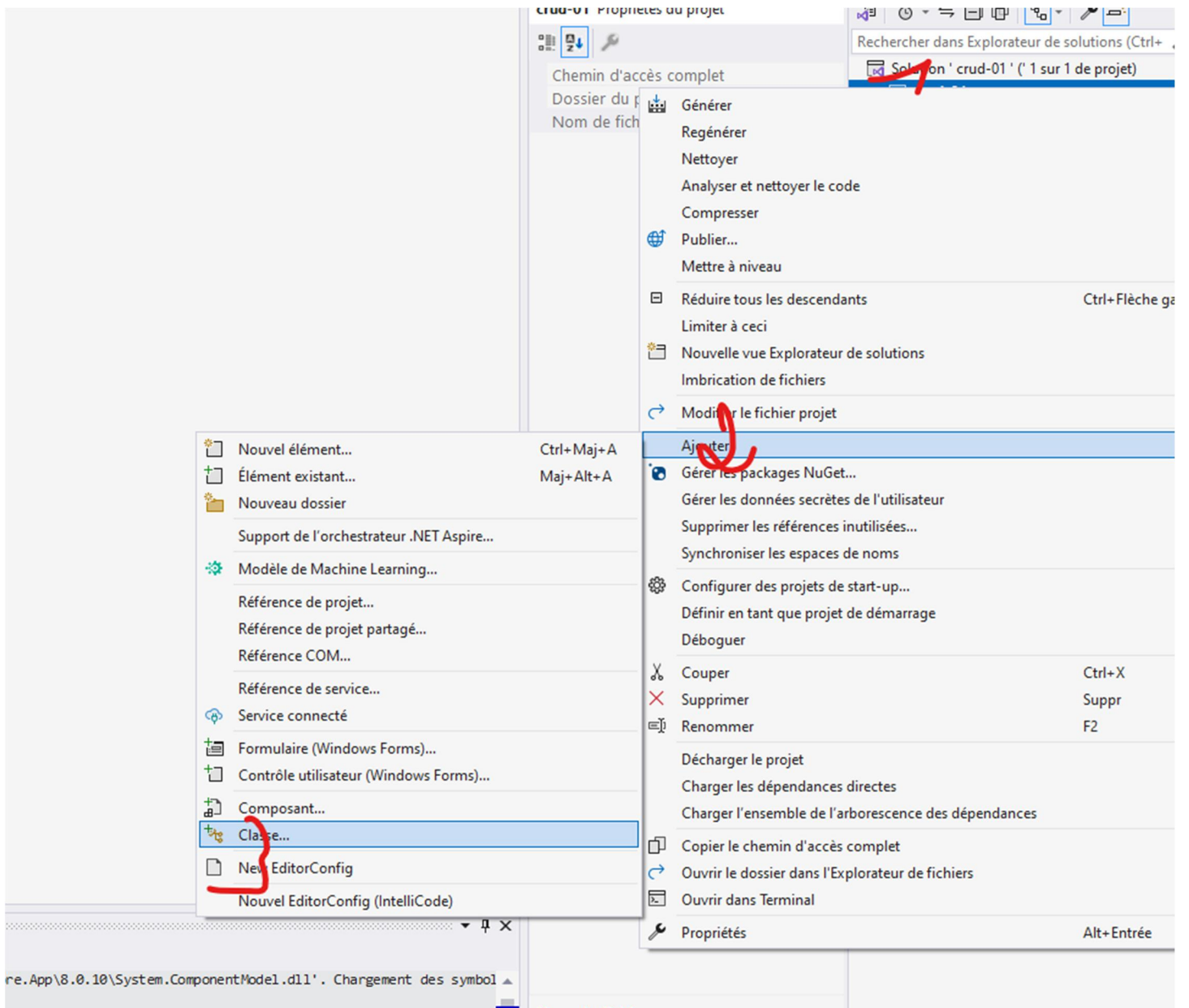
colonne(s)

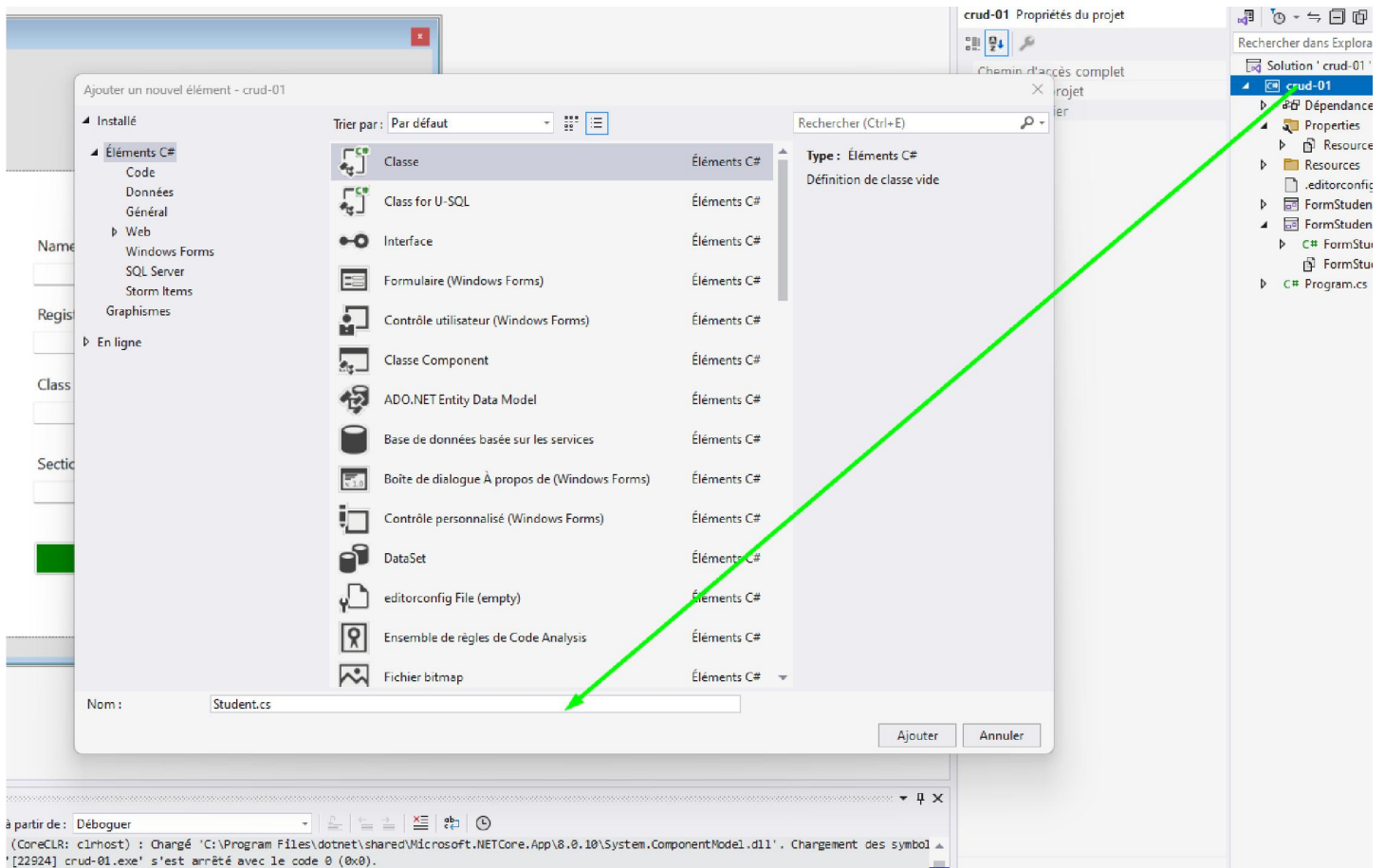
après Creat_at

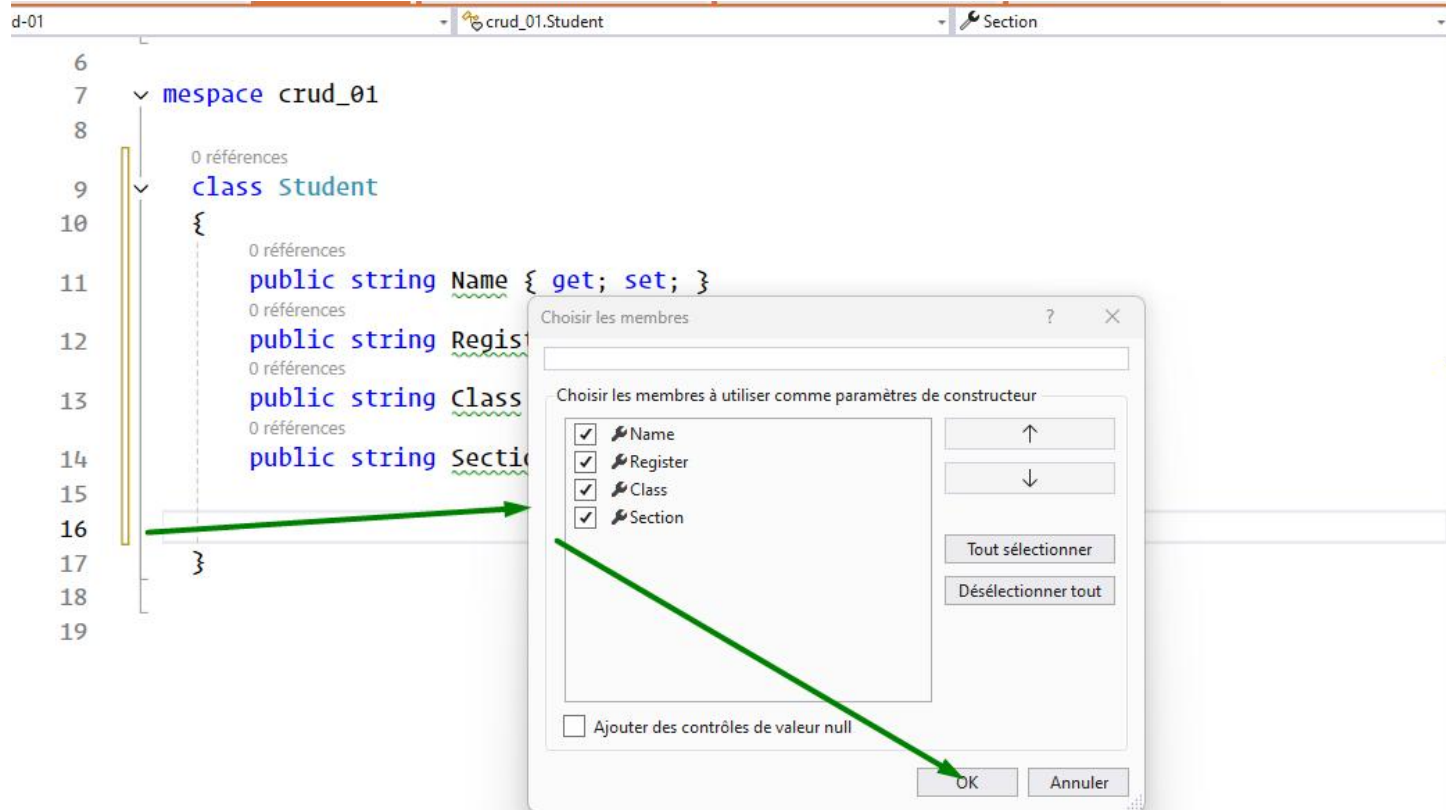
Exécuter

Index

Action	Nom de l'index	Type	Unique	Compressé	Colonne	Cardinalité	Interclassement	Null	Commentaire
Éditer Renommer Supprimer	PRIMARY	BTREE	Oui	Non	ID	0	A	Non	



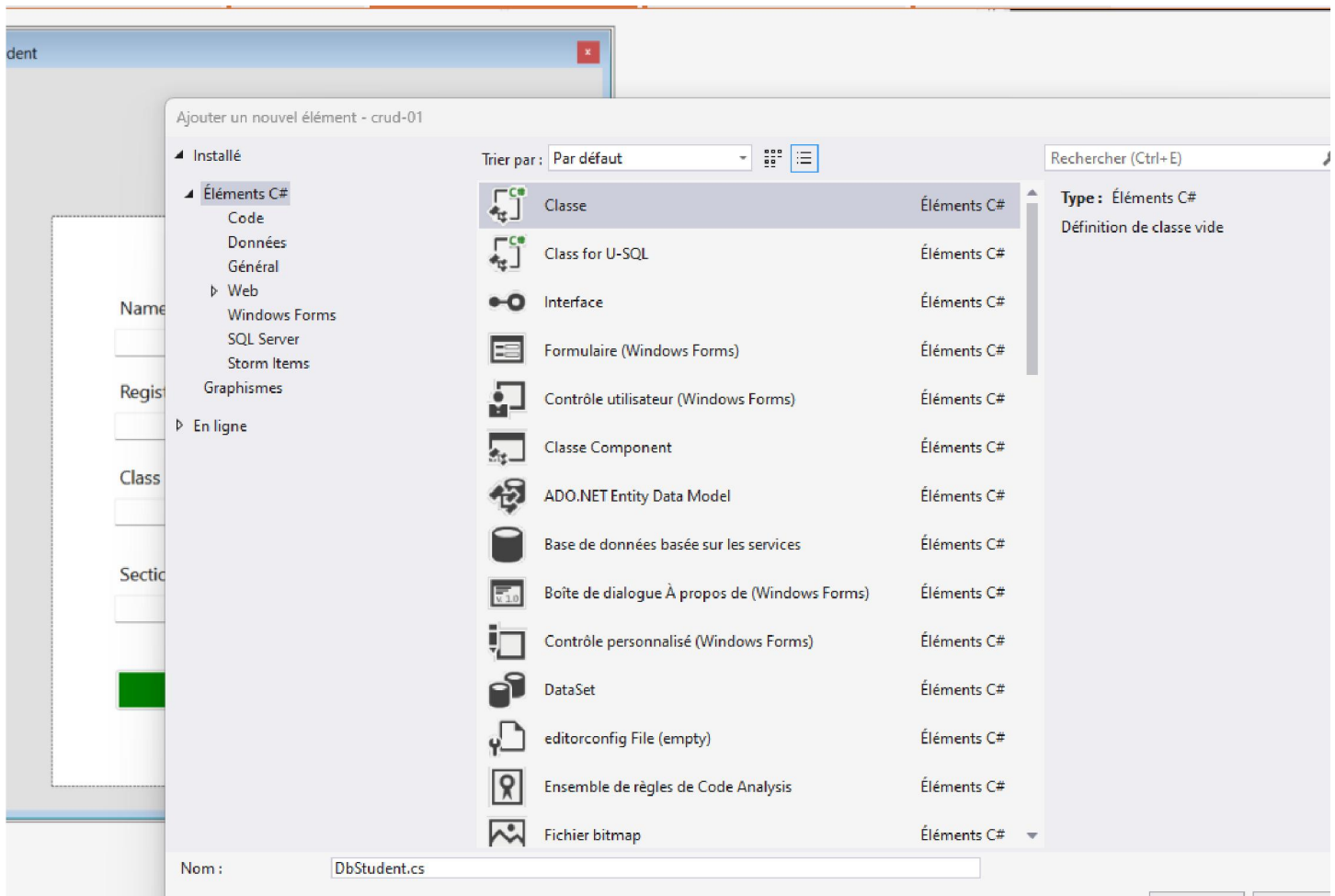




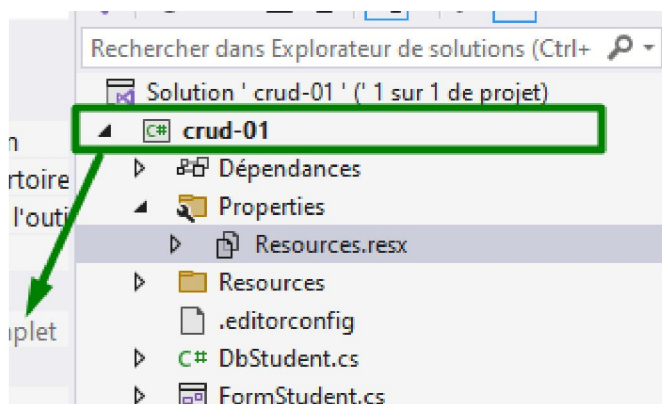
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

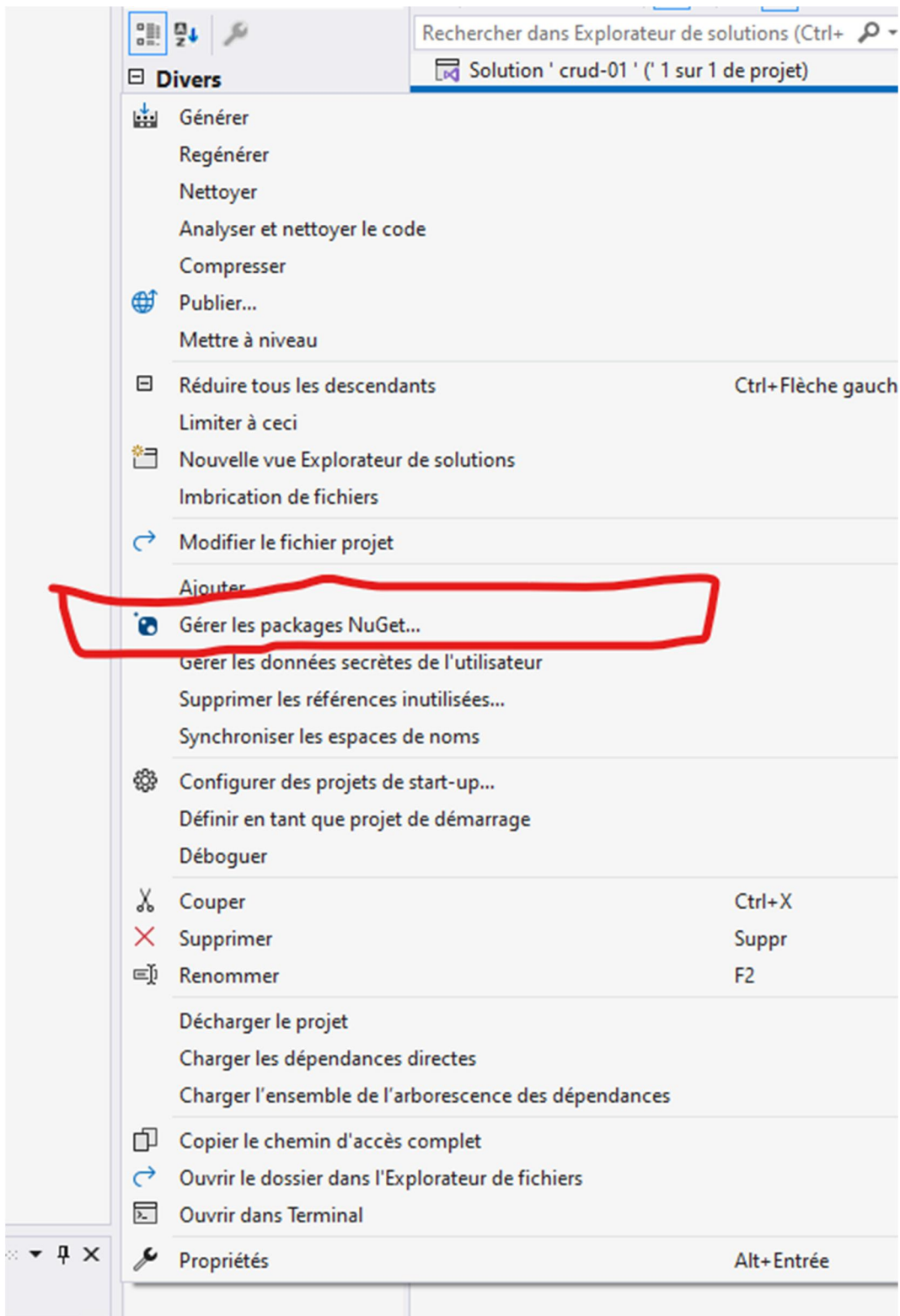
namespace crud_01
{
    class Student
    {
        public string Name { get; set; }
        public string Register { get; set; }
        public string Class { get; set; }
        public string Section { get; set; }

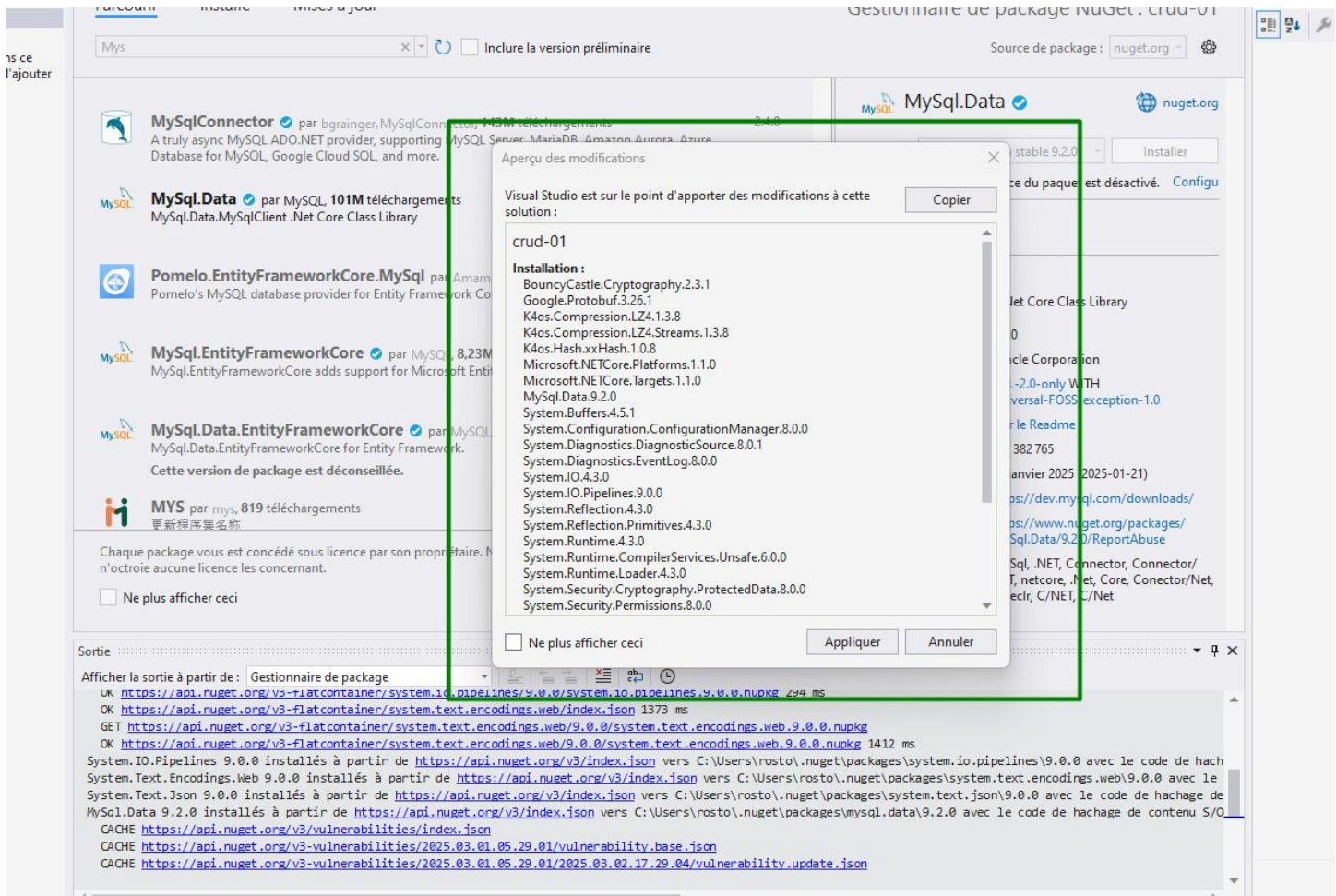
        public Student(string name, string register, string @class, string section)
        {
            Name = name;
            Register = register;
            Class = @class;
            Section = section;
        }
    }
}
```

Télécharger MySQL







<https://www.nuget.org/packages/MySQL.Data/9.2.0#show-readme-container>

```

MySQL.Data.MySqlClient.MySqlConnection myConnection;
    string myConnectionString;
    //set the correct values for your server, user, password and database name
    myConnectionString = "server=127.0.0.1;uid=root;pwd=12345;database=test";

    try
    {
        myConnection = new MySQL.Data.MySqlClient.MySqlConnection(myConnectionString);
        //open a connection
        myConnection.Open();

        // create a MySQL command and set the SQL statement with parameters
        MySqlCommand myCommand = new MySqlCommand();
        myCommand.Connection = myConnection;
        myCommand.CommandText = @"SELECT * FROM clients WHERE client_id = @clientId;";
        myCommand.Parameters.AddWithValue("@clientId", clientId);

        // execute the command and read the results
        using var myReader = myCommand.ExecuteReader()
        {
            while (myReader.Read())
            {
                var id = myReader.GetInt32("client_id");
                var name = myReader.GetString("client_name");
                // ...
            }
        }
    }
    myConnection.Close();
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}

```

Ce code est un exemple d'utilisation de la bibliothèque `MySQL.Data.MySqlClient` en C# pour interagir avec une base de données MySQL. Voici une explication détaillée de chaque section :

1. Déclarations des variables

```csharp

MySQL.Data.MySqlClient.MySqlConnection myConnection;

string myConnectionString;

...

- \*\*`myConnection`\*\* : Une variable de type `MySQLConnection` qui sera utilisée pour établir une connexion avec la base de données MySQL.

- `myConnectionString` : Une chaîne de connexion contenant les informations nécessaires pour se connecter à la base de données (adresse du serveur, identifiants, mot de passe, nom de la base).

---

### ## 2. Configuration de la chaîne de connexion

```
```csharp
```

```
myConnectionString = "server=127.0.0.1;uid=root;pwd=12345;database=test";
```

```
```
```

- `server=127.0.0.1` : L'adresse IP ou le nom d'hôte du serveur MySQL (ici, c'est un serveur local).

- `uid=root` : L'identifiant (nom d'utilisateur) pour se connecter à la base de données (ici, c'est `root`).

- `pwd=12345` : Le mot de passe associé à l'utilisateur `root`.

- `database=test` : Le nom de la base de données à laquelle on se connecte (ici, `test`).

---

### ## 3. Bloc `try-catch`

Le code est encapsulé dans un bloc `try-catch` pour gérer les exceptions qui pourraient survenir lors de la connexion ou des opérations sur la base de données.

---

### ## 4. Établir une connexion avec la base de données

```
```csharp
```

```
myConnection = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
```

```
myConnection.Open();
```

```
```
```

- Une instance de `MySqlConnection` est créée avec la chaîne de connexion.

- La méthode `Open()` ouvre une connexion avec le serveur MySQL.

---

### \*\*5. Création et configuration d'une commande SQL\*\*

```
```csharp
MySQLCommand myCommand = new MySQLCommand();
myCommand.Connection = myConnection;
myCommand.CommandText = @"SELECT * FROM clients WHERE client_id = @clientId;";
myCommand.Parameters.AddWithValue("@clientId", clientId);
...

```

- Une instance de `MySQLCommand` est créée pour exécuter une commande SQL.

- **`myCommand.Connection = myConnection`** : La commande est associée à la connexion ouverte.

- **Requête SQL** : La commande SQL sélectionne tous les enregistrements de la table `clients` où l'ID du client correspond à une valeur donnée.

- **Paramètre `@clientId`** : Un paramètre est utilisé pour éviter les attaques par injection SQL. La valeur du paramètre est définie avec `AddWithValue`.

6. Exécution de la commande et lecture des résultats

```
```csharp
using var myReader = myCommand.ExecuteReader()
{
 while (myReader.Read())
 {
 var id = myReader.GetInt32("client_id");
 var name = myReader.GetString("client_name");
 // ...
 }
}
...

```

- \*\*`ExecuteReader()`\*\* : Exécute la commande et retourne un objet `MySqlDataReader`, qui permet de lire les résultats ligne par ligne.

- **Lecture des résultats avec `Read()`** :
- La méthode `Read()` déplace le curseur sur chaque ligne retournée par la requête.
- Les colonnes des résultats sont accessibles via des méthodes comme `GetInt32()` pour un entier ou `GetString()` pour une chaîne.
- Dans cet exemple, les colonnes `client\_id` et `client\_name` sont extraites.

---

#### ### **\*\*7. Fermeture de la connexion\*\***

```
```csharp
myConnection.Close();
```
```

- La connexion à la base de données est fermée après l'exécution des opérations.

---

#### ### **\*\*8. Gestion des exceptions\*\***

```
```csharp
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```
```

- Si une erreur survient (par exemple, si le serveur MySQL est inaccessible ou si la requête échoue), une exception de type `MySqlException` est levée.
- Le message d'erreur est affiché dans une boîte de dialogue grâce à `MessageBox.Show`.

---

#### ### **\*\*Résumé global\*\***

Ce code permet :

1. De se connecter à une base de données MySQL.
2. D'exécuter une requête SQL paramétrée pour récupérer des données spécifiques.
3. De lire les résultats ligne par ligne.
4. De gérer les erreurs éventuelles grâce à un bloc `try-catch`.

---

### ### \*\*Améliorations possibles\*\*

1. \*\*Utilisation d'un bloc `using` pour la connexion\*\* :

- Cela garantit que la connexion est fermée automatiquement, même en cas d'exception.

```
```csharp
```

```
using (var myConnection = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString))
```

```
{
```

```
    myConnection.Open();
```

```
    // Opérations sur la base
```

```
}
```

```
```
```

2. \*\*Hashage des mots de passe\*\* :

- Il est déconseillé d'inclure directement des mots de passe en clair dans le code source.

3. \*\*Validation des entrées utilisateur\*\* :

- Bien que l'utilisation de paramètres (`@clientId`) protège contre les injections SQL, il est important de valider les entrées utilisateur avant leur utilisation.

```

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace crud_01
{
 public partial class FormStudentInfo : Form
 {
 public FormStudentInfo()
 {
 InitializeComponent();
 }
 private void BtnNew_Click (object sender, EventArgs e)
 {
 try
 {
 FormStudentok form = new FormStudentok(this);
 form.ShowDialog();
 }
 catch (Exception ex)
 {
 MessageBox.Show("Une erreur est survenue : " + ex.Message);
 }
 }
 }
}

```



```

using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace crud_01
{
 internal class DbStudent
 {
 public static MySqlConnection GetConnection()
 {
 string connectionString =
"datasource=localhost;port=3306;username=root;password=;database=student-dot-net";
 MySqlConnection connect = new MySqlConnection(connectionString);

 try
 {
 connect.Open();
 MessageBox.Show("Connexion a la DB avec success", "success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
 }
 catch (MySqlException ex)
 {
 MessageBox.Show("Erreur de connexion a la DB " + ex.Message, "Erreur",
MessageBoxButtons.OK, MessageBoxIcon.Error);
 }
 return connect;
 }

 public static void AddStudent(Student student)
 {
 MySqlConnection connect = GetConnection();
 string query = "INSERT INTO student (name, register, class, section) VALUES (@name,
@register, @class, @section)";
 MySqlCommand command = new MySqlCommand(query, connect);
 command.Parameters.AddWithValue("@name", student.Name);
 command.Parameters.AddWithValue("@register", student.Register);
 command.Parameters.AddWithValue("@class", student.Class);
 command.Parameters.AddWithValue("@section", student.Section);
 try
 {
 command.ExecuteNonQuery();
 MessageBox.Show("Etudiant ajoute avec success", "success", MessageBoxButtons.OK,
MessageBoxIcon.Information);
 }
 catch (MySqlException ex)
 {
 MessageBox.Show("Erreur d'ajout de l'etudiant " + ex.Message, "Erreur",
MessageBoxButtons.OK, MessageBoxIcon.Error);
 }
 connect.Close();
 }
 }
}

```

```

 }
}

```

```

using System;
using System.Formats.Asn1;
using System.Windows.Forms;
using System.Xml.Linq;

namespace crud_01
{
 public partial class FormStudentok : Form
 {
 private readonly FormStudentInfo _parent;

 public FormStudentok(FormStudentInfo parent)
 {
 InitializeComponent();
 _parent = parent;
 }

 public void Clear()
 {
 txtName.Text = txtRegister.Text = txtClass.Text = txtSection.Text = string.Empty;
 }

 private void btnSave_Click(object sender, EventArgs e)
 {
 if (txtName.Text.Trim().Length < 3)
 {
 MessageBox.Show("Le nom doit contenir au moins 3 caractères", "Erreur",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
 return;
 }
 if (txtRegister.Text.Trim().Length < 3)
 {
 MessageBox.Show("Le matricule doit contenir au moins 3 caractères", "Erreur",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
 return;
 }
 if (txtClass.Text.Trim().Length < 3)
 {
 MessageBox.Show("La classe doit contenir au moins 3 caractères", "Erreur",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
 return;
 }
 if (txtSection.Text.Trim().Length < 3)
 {
 MessageBox.Show("La section doit contenir au moins 3 caractères", "Erreur",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
 return;
 }

 if (btnSave.Text == "Save") // Correction ici
 {

```

```

 Student student = new Student(txtName.Text, txtRegister.Text, txtClass.Text,
txtSection.Text);
 DbStudent.AddStudent(student);

 Clear();
 }

}
}
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace crud_01
{
 class Student
 {
 public string Name { get; set; }
 public string Register { get; set; }
 public string Class { get; set; }
 public string Section { get; set; }

 public Student(string name, string register, string @class, string section)
 {
 Name = name;
 Register = register;
 Class = @class;
 Section = section;
 }
 }
}

```