**TRƯỜNG ĐẠI HỌC**

**KINH TẾ KỸ THUẬT CÔNG NGHIỆP**

**KHOA CNTT**

**LẬP TRÌNH XỬ LÝ DỮ LIỆU NÂNG CAO**

**TÌM HIỂU VỀ CÔNG NGHỆ ADO.NET QUẢN LÝ DỮ LIỆU THEO MÔ HÌNH BA LỚP**

***Giáo viên hướng dẫn*: Hoàng Thị Minh Châu**

|  |  |
| --- | --- |
| Nhóm thực hiện: | **ITA2 - K11**   1. Nguyễn Khắc Mạnh 2. Đỗ Văn Lập 3. Vũ Minh Hiếu 4. Nguyễn Xuân Chiến 5. Phạm Bá Hùng 6. Nguyễn Văn Khoa |

Hà Nội, 10/2020

# BẢNG PHÂN CÔNG NHIỆM VỤ

|  |  |  |
| --- | --- | --- |
| Thành viên | Nhiệm vụ | Chức vụ |
| 1. Nguyễn Khắc Mạnh | Code và demo về mô hình 3 lớp |  |
| 1. Đỗ Văn Lập |  |  |
| 1. Vũ Minh Hiếu |  |  |
| 1. Nguyễn Xuân Chiến |  |  |
| 1. Phạm Bá Hùng |  |  |
| 1. Nguyễn Văn Khoa |  |  |

# CHƯƠNG 1: GIỚI THIỆU VỀ CÔNG NGHỆ ADO.NET

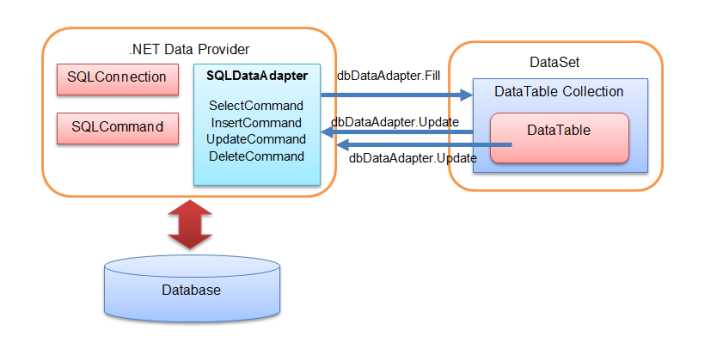
## ADO.Net là gì ?

## Giới thiệu:

ADO.NET là một bộ thư viện hướng đối tượng (OOP) cho phép tương tác với dữ liệu nguồn. Thông thường  thì dữ liệu nguồn là một cơ sở dữ liệu (Database), nhưng củng có thể là file text, exel hoặc XML. Ở đây ta chỉ xét ADO.NET làm việc với database.

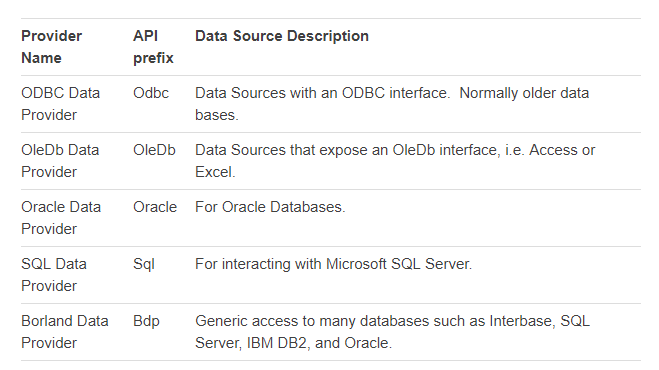
ADO.NET là một kỹ thuật .NET để thao tác với nguồn dữ liệu. Nó là môt tập hợp các lớp  nằm trong bộ thư viện lớp cơ sở của .NET Framework.

## Data Provider

ADO.NET cho phép tương tác với các loại dữ liệu và các kiễu database. Với mỗi loại dữ liệu lại cần một phương thức khác nhau để có thể truy xuất. Cần có một thư viên thống nhất để tương tác với nguồn dữ liệu, đó chính là lý do ADO.NET được tạo ra.

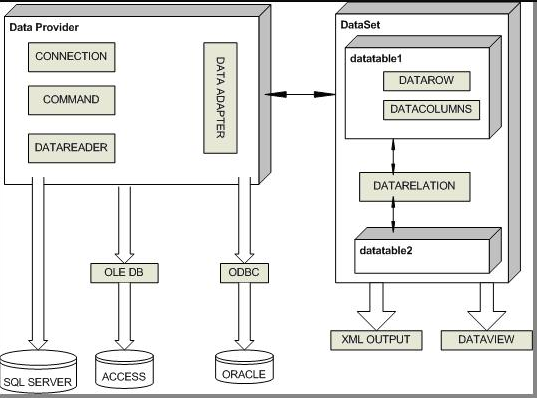
ADO.NET cung cấp một cách thức chung để tương tác với nguồn dữ liệu, với mỗi loại dữ liệu bạn cần phải sử dụng một thư viện khác nhau. Các thư viện này được gọi là Data Provider.

Bảng sau liệt kê các Data Provider  phổ biến:



## Các thành phần của ADO.NET

ADO.NET có 2 thành phần chính: Data Provider và DataSets



Phần này giới thiệu một vài đối tượng chính thường sử dụng.

### SqlConnection

Là một kết nối tới database. Kết nối giúp xác định Database server, database name, user name, password và các tham số cần thiết để kết nối tới database.

Đối tượng Connection thường được dùng với đối tượng Command.

### SqlCommand

Cho biết hành động mà ta muốn thực hiện với database. Đối tượng command dùng để thực thi SQL queries, câu lênh hoặc lưu trữ các thủ tục.

Một đối tượng Command dùng một đối tượng Connection để xác định cái database mà nó muốn nói chuyện.

Ta có thể dùng một đối tượng Command riêng lẻ để thực thi lệnh trực tiếp hoặc gắn vào một SqlDataAdapter.

### SqlDataAdapter

Là cầu nối trung gian giữa dataset với data source.

### SqlDataReader

Cung cấp một data stream tới kết quả.

### SqlError

lưu trữ thông tin về lỗi và cảnh báo.

### SqlException

Các ngoại lệ trong trường hợp SQL Server lỗi và cảnh báo.

### SqlParameter

tham số cho Command

### DataSet

Là một thể hiện của dữ liệu trong bộ nhớ. Giống như một database nhỏ.

DataSet cho phép lưu trử và chỉnh sữa dữ liệu tại ‘local cache‘

### Đối tượng DataTable và DataColumn

Ta có thể viết mã C# để tạo ra đối tượng DataTable hay nhận về từ kết quả của câu truy vấn đến cơ sở dữ liệu. DataTable có một số thuộc tính dùng chung ( public ) như thuộc tính Columns, từ thuộc tính này ta có thể truy cập đến đối tượng DataColumnsCollection thông qua chỉ mục hay tên của cột để nhận về các đối tượng DataColumn thích hợp, mỗi DataColumn tương ứng với một cột trong một bảng dữ liệu.

DataTable dt = new DataTable("tenBang"); DataColumn dc = dt.Columns["tenCot"];

### Đối tượng DataRelation

Ngoài tập các đối tượng DataTable được truy cập thông qua thuộc tính Tables, DataSet còn có một thuộc tính Relations. Thuộc tính này dùng để truy cập đến đối tượng DataRelationCollection thông qua chỉ mục hay tên của quan hệ và sẽ trả về đối tượng DataRelation tương ứng.

DataSet ds = new DataSet("tenDataSet"); DataRelation dre = ds.Relations["tenQuanHe"];

### Các bản ghi ( Rows )

Tương tự như thuộc tính Columns của đối tượng DataTable, để truy cập đến các dòng ta cũng có thuộc tính Rows. ADO. NET không đưa ra khái niệm RecordSet, thay vào đó để duyệt qua các dòng ( Row ), ta có thể truy cập các dòng thông qua thuộc tính Rows bằng vòng lặp foreach.

### Đối tượng SqlConnection và SqlCommand

Đối tượng SqlConnection đại diện cho một kết nối đến cơ sở dữ liệu, đối tượng này có thể được dùng chung cho các đối tượng SqlCommand khác nhau. Đối tượng SqlCommand cho phép thực hiện một câu lệnh truy vấn trực tiếp : như SELECT, UPDATE hay DELETE hay gọi một thủ tục (Store Procedure) từ cơ sở dữ liệu.

### Đối tượng DataAdapter

ADO.NET dùng DataAdapter như là chiếc cầu nối trung gian giữa DataSet và DataSource ( nguồn dữ liệu ), nó lấy dữ liệu từ cơ sở dữ liệu sau đó dùng phương Fill() để đẩy dữ liệu cho đối tượng DataSet. Nhờ đối tượng DataAdapter này mà DataSet tồn tại tách biệt, độc lập với cơ sở dữ liệu và một DataSet có thể là thể hiện của một hay nhiều cơ sở dữ liệu. Ví dụ :

//Tạo đối tượng SqlDataAdapter

SqlDataAdapter sda = new SqlDataAdapter();

// cung cấp cho sda một SqlCommand và SqlConnection ...

// lấy dữ liệu ...

//tạo đối tượng DataSet mới

DataSet ds = new DataSet("tenDataSet");

//Đẩy dữ liệu trog sda vào ds bằng hàm Fill();

sda.Fill(ds);

### Trình cung cấp dữ liệu (.NET Data Providers)

.NET Framework hỗ trợ hai trình cung cấp dữ liệu là SQL Server .NET Data Provider ( dành cho phiên bản SQL Server 7.0 của Microsoft trở lên ) và OLE DB .NET Data Provider ( dành cho các hệ quản trị cơ sở dữ liệu khác ) để truy cập vào cơ sở dữ liệu.

So sánh SQL Server .NET Data Provider và the OLE DB .NET Data Provider

SQL Server .NET Data Provider có các đặc điểm :

* Dùng nghi thức riêng để truy cập cơ sở dữ liệu
* Truy xuất dữ liệu sẽ nhanh hơn và hiệu quả hơn do không phải thông qua lớp OLE DB Provider hay ODBC
* Chỉ được dùng với hệ quản trị cơ sở dữ liệu SQL Server 7.0 trở lên.
* Được Mircrosoft hỗ trợ khá hoàn chỉnh.

OLE DB .NET Data Provider có các đặc điểm :

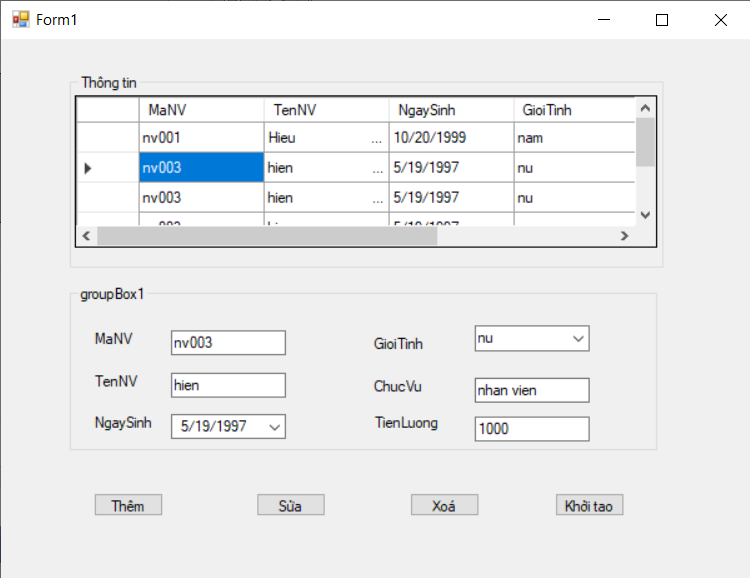
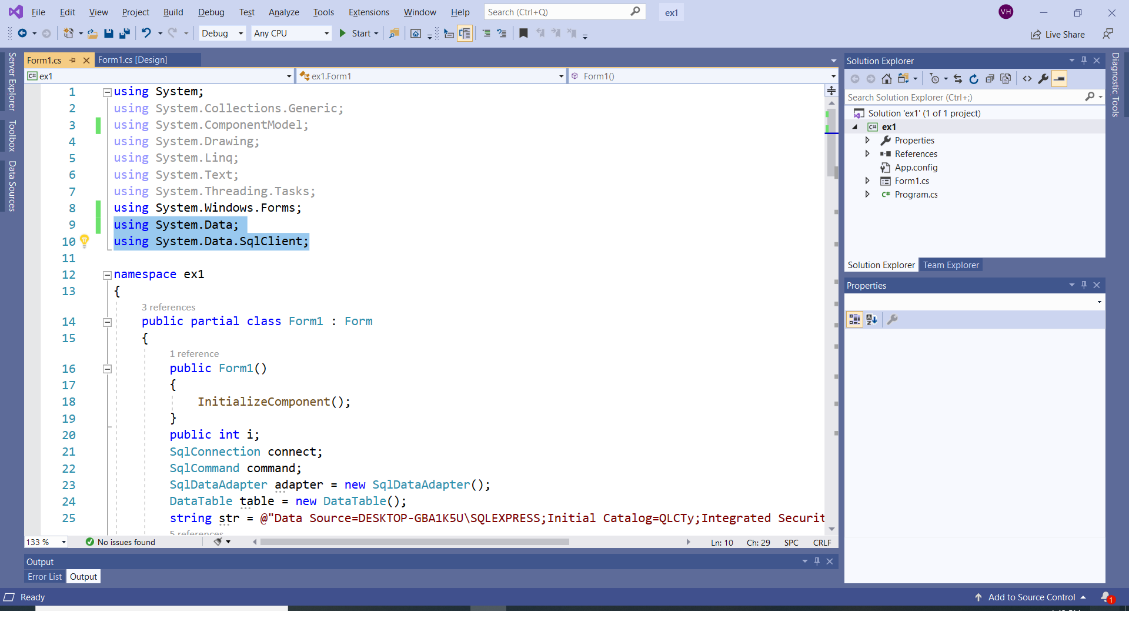
* Phải thông qua 2 lớp vì thế sẽ chậm hơn
* Thực hiện được các dịch vụ “Connection Pool”
* Có thể truy cập vào mọi Datasource có hỗ trợ OLE DB Provider thích hợp

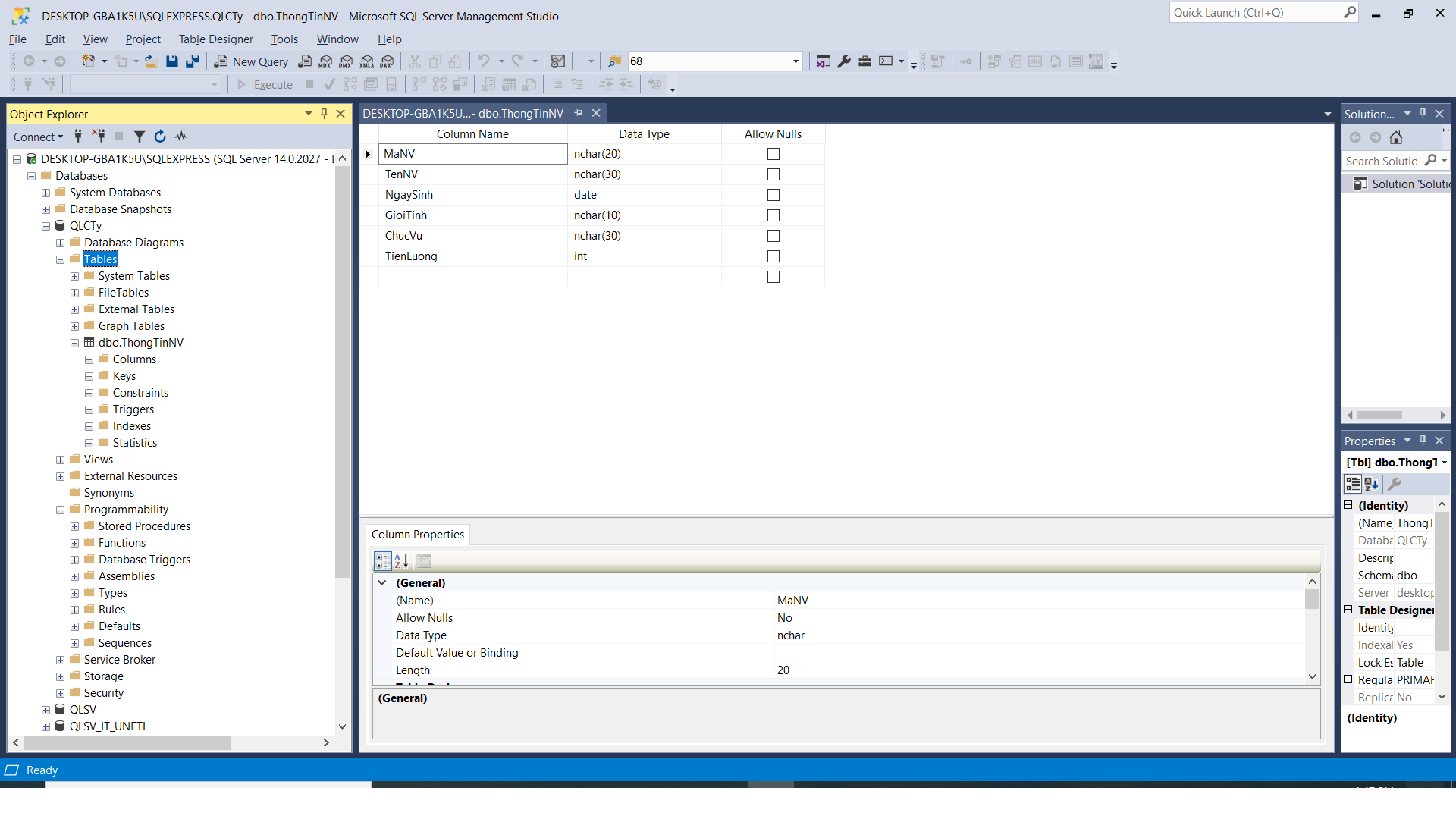
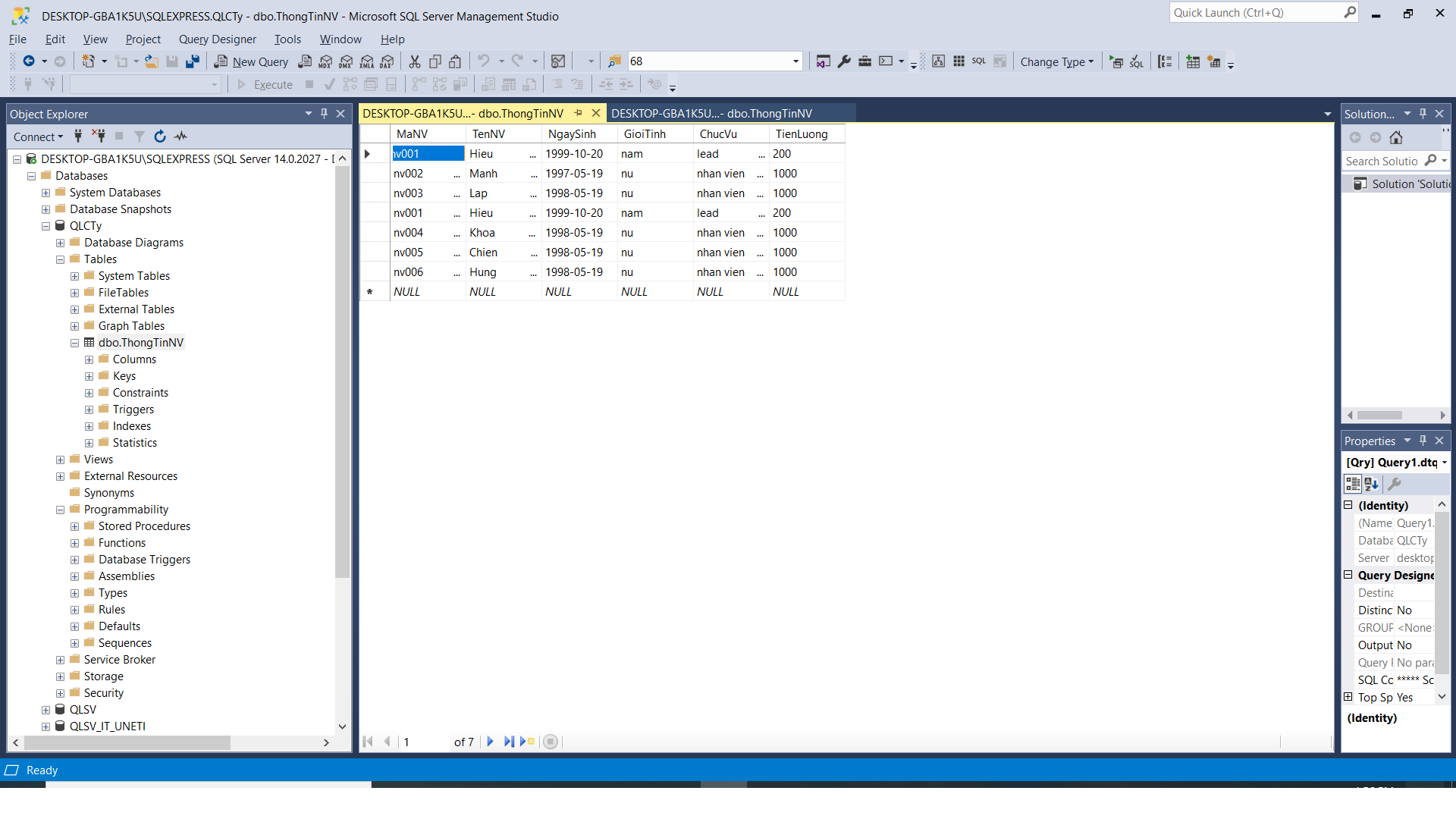
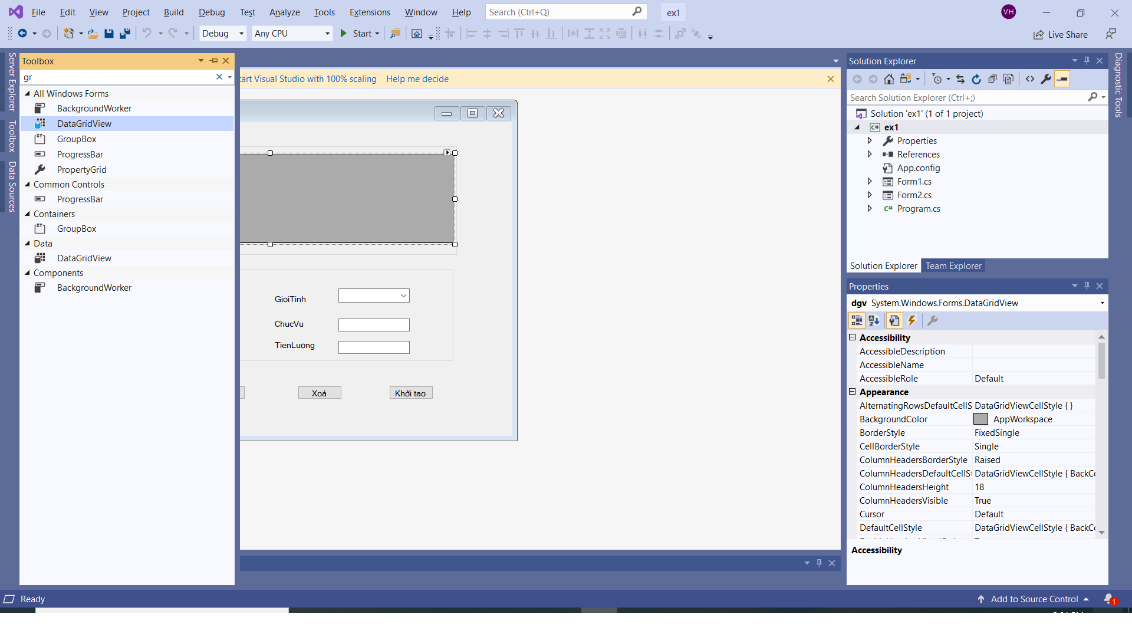
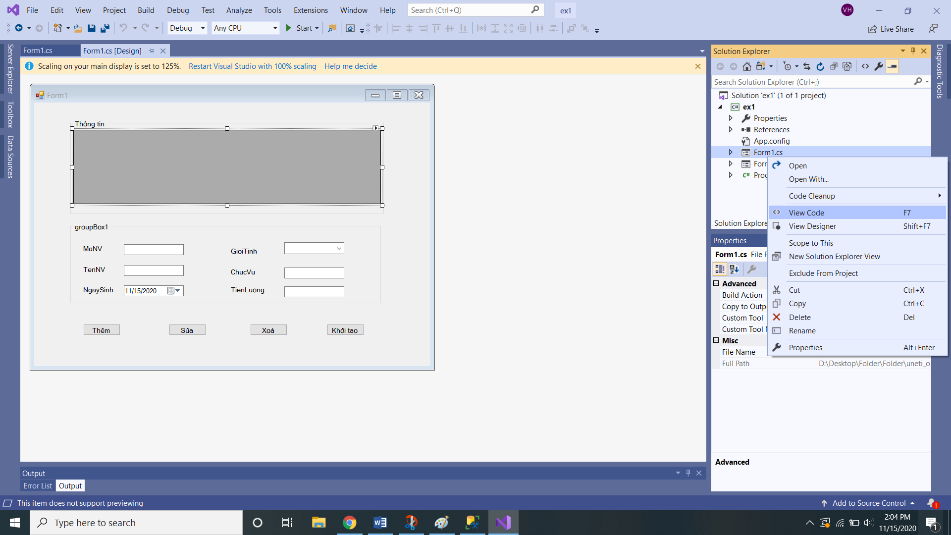
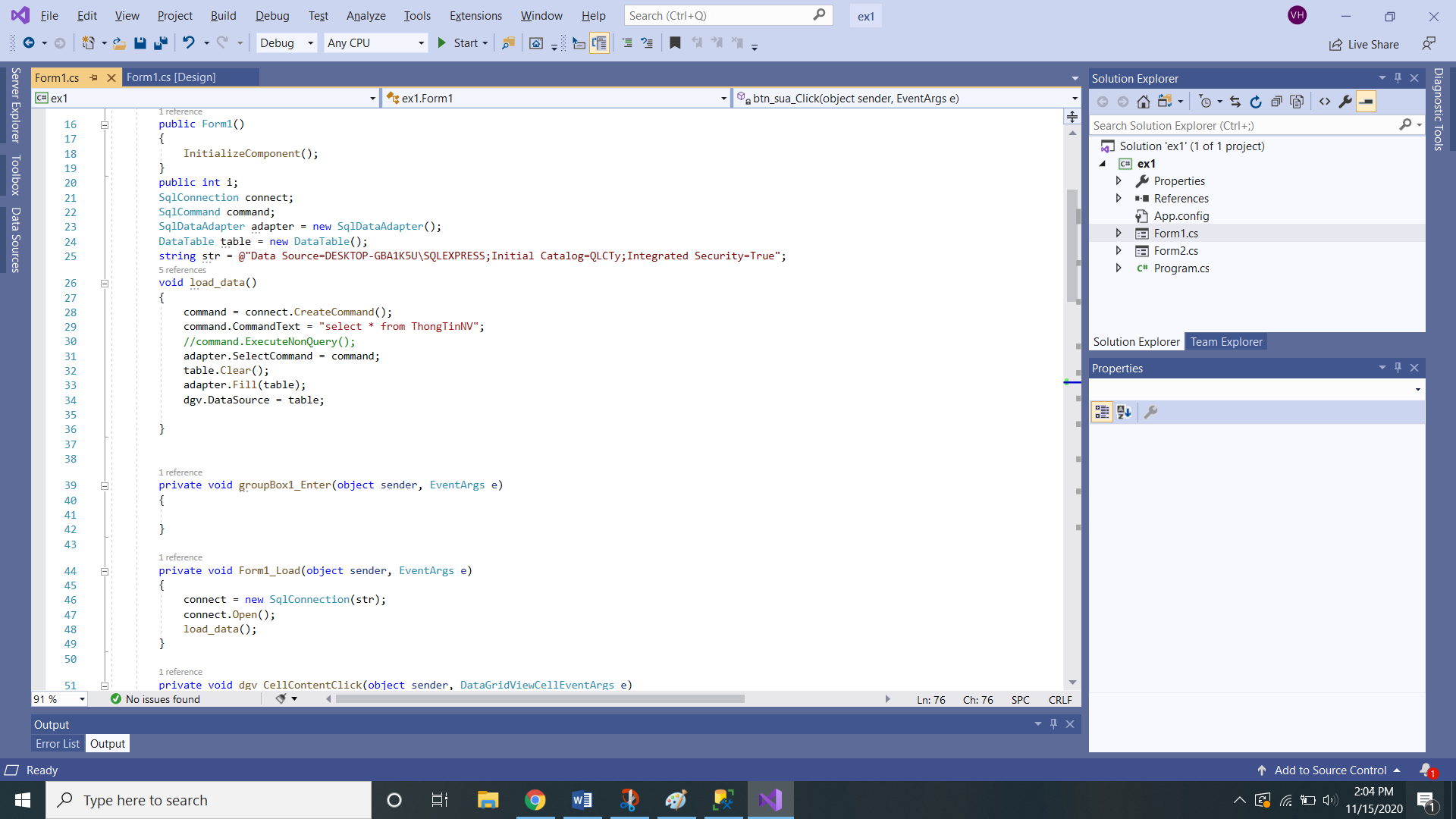
# CHƯƠNG 2: BÀI TẬP HIỂN THỊ DỮ LIỆU BẰNG DATAGIRDVIEW, GIRD VIEW

**GridView là một trong những control rất quan trọng khi thiết kế web và**[**lập trình ASP.NET**](https://aptech.vn/kien-thuc-tin-hoc/huong-dan-gui-mail-trong-asp-net.html)**. Control này cho phép lập trình viên dễ dàng hiển thị dữ liệu và các thông tin quan trọng theo dạng bảng gần giống với ứng dụng văn phòng quen thuộc Microsoft Excel.**

**DataGridview là** control dùng để hiển thị dữ liệu lên trên giao diện

DataTable đại diện cho đối tượng cung cấp data còn **dataGridView là** đối tượng để hiển thị danh sách dữ liệu dưới dạng grid.

**Hướng dẫn Kết nối CSDL và hiển thị DataGridview trong C# Winform**  
  
Ảnh DEMO:  
  
  
  
**Thêm 2 thư viện:**  
using System.Data;  
using System.Data.SqlClient;  
  


**Tạo database:**  
  
Sử dụng SQL Server  
  
Tiếp theo: Tạo database với tên **QLCTy** và bảng table là **ThongTinNV**rồi điền thông tin như hình  
  
  
  
Nhập CSDL  
  
  
 **Mở visual studio**  
Tạo Project mới-> **New -> Project -> Visual C# -> Windows Forms Application**  
Tại **Form** -> Kéo **Datagridview** vào form  
  
  
  
  
  
  
  
Đặt name cho **Datagridview**   
  
Sử dụng DataTable để lưu dữ liệu  
  
**Đoạn code kết nối CSDL**  
  
  
  


CODE:

Select All

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;

namespace ex1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

public int i;

SqlConnection connect;

SqlCommand command;

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

string str = @"Data Source=DESKTOP-GBA1K5U\SQLEXPRESS;Initial Catalog=QLCTy;Integrated Security=True";

void load\_data()

{

command = connect.CreateCommand();

command.CommandText = "select \* from ThongTinNV";

//command.ExecuteNonQuery();

adapter.SelectCommand = command;

table.Clear();

adapter.Fill(table);

dgv.DataSource = table;

}

private void Form1\_Load(object sender, EventArgs e)

{

connect = new SqlConnection(str);

connect.Open();

load\_data();

}

private void dgv\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

//tb\_ma.ReadOnly = true;

int i;

i = dgv.CurrentRow.Index;

tb\_ma.Text = dgv.Rows[i].Cells[0].Value.ToString();

tb\_ten.Text= dgv.Rows[i].Cells[1].Value.ToString();

dt\_ns.Text= dgv.Rows[i].Cells[2].Value.ToString();

cb\_gt.Text= dgv.Rows[i].Cells[3].Value.ToString();

tb\_chuc.Text= dgv.Rows[i].Cells[4].Value.ToString();

tb\_luong.Text= dgv.Rows[i].Cells[5].Value.ToString();

}

private void btn\_them\_Click(object sender, EventArgs e)

{

command = connect.CreateCommand();

command.CommandText = "insert into ThongTinNV values('"+tb\_ma.Text+"','"+tb\_ten.Text+"','"+dt\_ns.Text+"','"+cb\_gt.Text+"','"+tb\_chuc.Text+"','"+tb\_luong.Text+"')";

command.ExecuteNonQuery();

load\_data();

}

command = connect.CreateCommand();

command.CommandText = "update ThongTinNV set TenNV= '" + tb\_ten.Text + "', TienLuong= '" + tb\_luong.Text+"', ChucVu= '"+tb\_chuc.Text+"' where MaNV='"+tb\_ma.Text+"'";

command.ExecuteNonQuery();

load\_data();

}

private void btn\_xoa\_Click(object sender, EventArgs e)

{

command = connect.CreateCommand();

command.CommandText = "delete from ThongTinNV where MaNV='" + tb\_ma.Text + "' ";

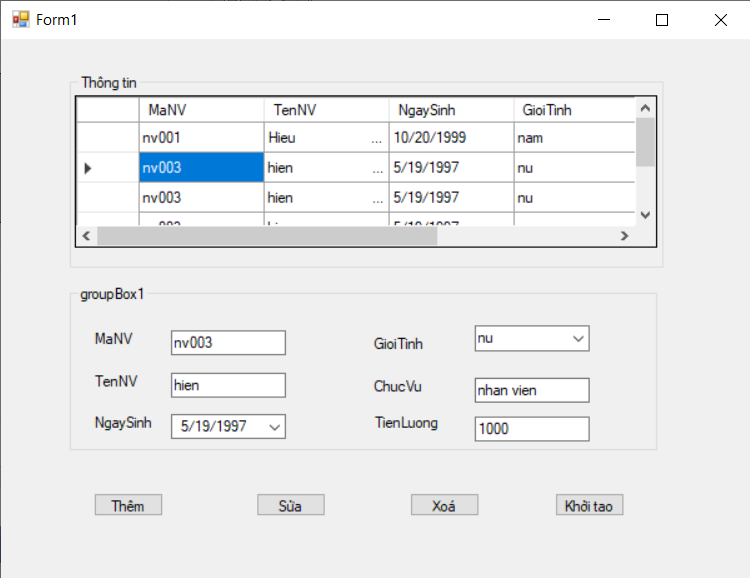
command.ExecuteNonQuery();

load\_data();

}

}

}

Kết quả:  
  


# CHƯƠNG 3: BÀI TẬP VỀ QUẢN LÝ DỮ LIỆU THEO MÔ HÌNH BA LỚP

Đối với lập trình hiện đại, mọi thứ đều đi nhiều lớp, tương tự như C#, cũng có cả **mô hình 3 lớp C#** để chúng ta thực thi áp dụng.

Ở C# chúng ta gọi là mô hình 3 lớp aka 3 Layers. Nó khá là nổi tiếng với sinh viên VN đang học C# (một số trường sẽ là Lập trình .NET, lập trình C#, lập trình ứng dụng).

### **Mục lục:**

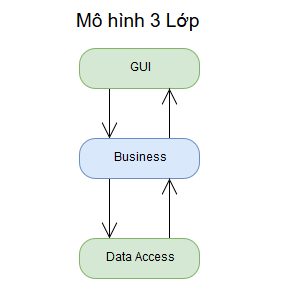
1. Giới thiệu về mô hình 3 lớp.
2. Cách tạo project và liên kết 3 lớp.
3. Xây dựng DTO
4. Xây dựng Data Access
5. Xây dựng Business (BUS)
6. Xây dựng GUI
7. Lời kết và Source Code mẫu

### **Cấu tạo của C# – Mô hình 3 lớp đơn giản:**

Gồm 3 lớp, đó là:

* **GUI** Layer: Lớp này là lớp hiển thị giao diện và các chức năng để người dùng cuối sử dụng.
* **Business** (BUS) Layer: Đây là lớp nhận các yêu cầu từ lớp GUI và truy xuất lên lớp Data để lấy thông tin và trả về GUI.
* **Data Access** Layer: Lớp này là lớp để truy xuất với CSDL, chỉ duy nhất lớp này được làm việc với database.
* (Ko cần thiết) DTO Layer: Lớp này chỉ là phụ thôi, đây là lớp định nghĩa các table trong database của bạn, định nghĩa cột của nó cũng như để ta gán data khi query lấy dữ liệu. Các bạn có thể hiểu nôm na là 1 dạng cơ bản ORM (Object Relation Mapping).

Đây là cách hoạt động của mô hình 3 lớp:



Nhìn sơ qua thì nó khá là giống MVC bên web nhỉ? Business như là Controller :D, GUI là View và Data Access là Model.

### **Lợi thế của mô hình 3 lớp:**

* Phân loại rõ ràng các lớp có các nhiệm vụ khác nhau. Từ đó ta có thể quản lý và maintain project tốt hơn.
* Dễ dàng phân loại các hành động tại Business.
* Dễ dàng phân loại các hàm truy xuất tại Database, phân loại hàm theo table,…
* Ứng dụng được cho các project lớn ở bên ngoài.
* …

### **Lưu ý khi xây dựng mô hình 3 lớp:**

* Cần một solution riêng cho project.
* Cần 3 project khác nhau để làm nên 3 lớp, tên **Project** đặt như sau:
  + Lớp **GUI**: GUI\_\* (VD: GUI\_QuanLy)
  + Lớp **Business**: BUS\_\* (VD: BUS\_QuanLy)
  + Lớp **Data Access**: DAL\_\* (VD: DAL\_QuanLy)
  + Lớp **DTO**: DTO\_\* (VD: DTO\_QuanLy)
* Bên trong 3 lớp như trên các file đặt cần có các tiền tố như sau:  
  Ví dụ mình có một table tên là **ThanhVien**

* + Lớp **GUI**: GUI\_\* (VD: GUI**\_ThanhVien**)
  + Lớp **Business**: BUS\_\* (VD: GUI**\_ThanhVien**)
  + Lớp **Data Access**: DAL\_\* (VD: GUI**\_ThanhVien**)
  + Lớp **DTO**: DTO\_\* (VD: DTO**\_ThanhVien**)

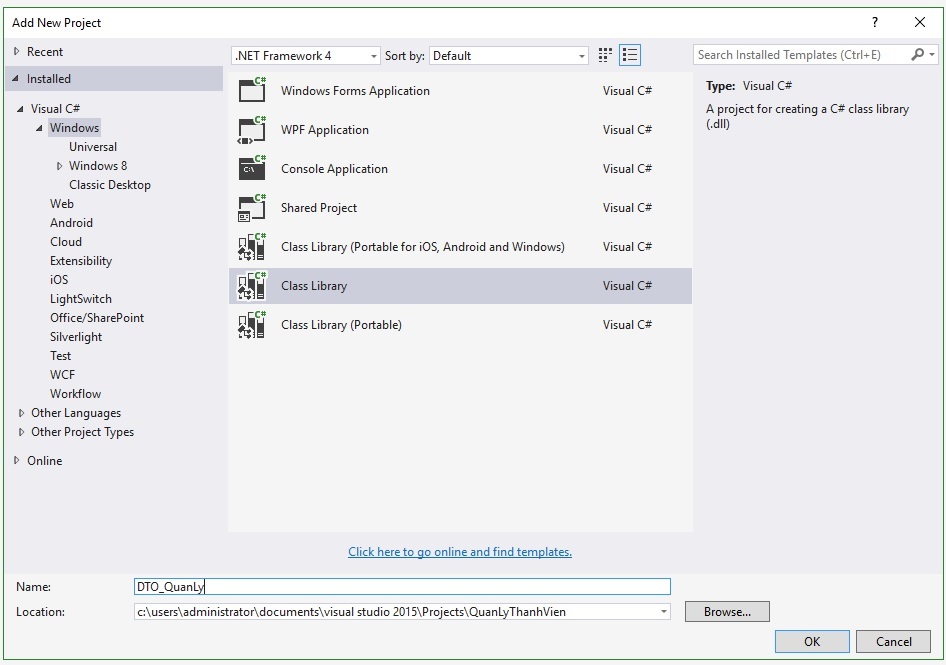
Như các bạn đã thấy tên Table liên quan mật thiết tới cách đặt tên file nhé

### **Liên kết 3 lớp**

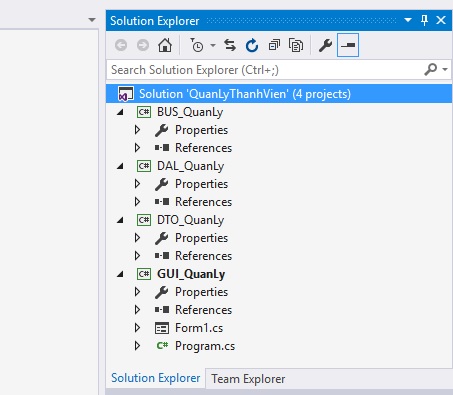
Như các bạn đã nhìn tại sơ đồ ở trang 1 thì Mô hình 3 lớp hoạt động như sau:

* GUI liên kết tới dc**Business Layer** và **DTO**.
* Business Layer liên kết tới được**Data Access** và **DTO**.
* Data Access chỉ liên kết tới **DTO**.

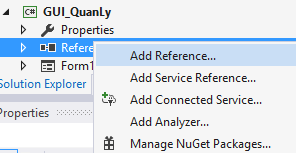
Bây giờ chúng ta bắt đầu tạo, đối với 3 Project DTO, Business và Data Access chúng ta tạo theo **Class Library** nhé



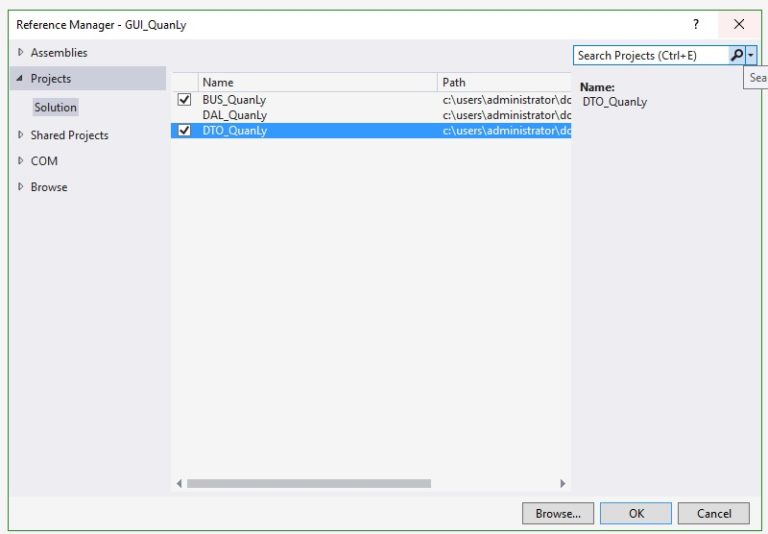
Và khi tạo xong, ta sẽ có 4 project như sau(Có mấy file class1.cs xóa thoải mái nhé.



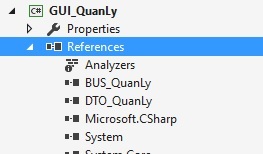
Và bây giờ chúng ta sẽ liên kết, làm project GUI trước nhé. Đầu tiên ta chọn chuột fải vào **References => Add Reference**

****

Một cái bảng hiện ra, tại mục Project ta sẽ chọn 2 lớp mà lớp **GUI** có thể liên kết tới là **BUS** và **DTO**:



Cứ OK là Reference sẽ được thêm vào GUI, chúng ta có thể kiểm tra bằng cách mở **References** ra:



Như bên hình thì mình đã thêm thành công, và các bạn làm tương tự đối với **BUS** và **DAL** theo mình nói bên trên nhé.

Thế là bạn đã liên kết xong cho mô hình 3 lớp.

### **Xây dựng mô hình 3 lớp: Xây dựng DTO**

### **Cơ sở dữ liệu:**

CSDL mình sẽ xài table **ThanhVien** như sau:

CREATE TABLE THANHVIEN (

TV\_ID INT NOT NULL PRIMARY KEY IDENTITY,

TV\_NAME NVARCHAR(30) NOT NULL,

TV\_PHONE VARCHAR(11) NOT NULL,

TV\_EMAIL VARCHAR(50) NOT NULL

)

### **Xây dựng DTO**

Mình sẽ tạo file **DTO\_ThanhVien**.cs (Class file), về cơ bản thì các cột của table ThanhVien của mình ra sao thì mình tạo 1 class y hệt vậy kèm get/set và 2 hàm khởi tạo.

**DTO\_ThanhVien**.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace DTO\_QuanLy

{

public class DTO\_ThanhVien

{

private int \_THANHVIEN\_ID;

private string \_THANHVIEN\_NAME;

private string \_THANHVIEN\_PHONE;

private string \_THANHVIEN\_EMAIL;

/\* ======== GETTER/SETTER ======== \*/

public int THANHVIEN\_ID

{

get

{

return \_THANHVIEN\_ID;

}

set

{

\_THANHVIEN\_ID = value;

}

}

public string THANHVIEN\_NAME

{

get

{

return \_THANHVIEN\_NAME;

}

set

{

\_THANHVIEN\_NAME = value;

}

}

public string THANHVIEN\_PHONE

{

get

{

return \_THANHVIEN\_PHONE;

}

set

{

\_THANHVIEN\_PHONE = value;

}

}

public string THANHVIEN\_EMAIL

{

get

{

return \_THANHVIEN\_EMAIL;

}

set

{

\_THANHVIEN\_EMAIL = value;

}

}

/\* === Constructor === \*/

public DTO\_ThanhVien()

{

}

public DTO\_ThanhVien(int id, string name, string phone, string email)

{

this.THANHVIEN\_ID = id;

this.THANHVIEN\_EMAIL = email;

this.THANHVIEN\_NAME = name;

this.THANHVIEN\_PHONE = phone;

}

}

}

### **Xây dựng mô hình 3 lớp: Xây dựng Data Access**

Tại sao mình lại xây dựng Data Access trước? Đơn giản là đây là lớp mà ta xử lý bên database, làm trước thì design GUI xong chỉ việc bỏ vào sử dụng thôi.

Điều quan trọng đầu tiên, chúng ta cần tạo class **DBConnect.cs** với nội dung như sau:

using System.Data.SqlClient;

namespace DAL\_QuanLy

{

public class DBConnect

{

protected SqlConnection \_conn = new SqlConnection("Data Source=ADMINISTRATORSQLEXPRESS;Initial Catalog=ThanhVien;Integrated Security=True");

}

}

Chúng ta sẽ tạo **SqlConnection** và khởi tạo luôn, sau này các**class DAL** chúng ta chỉ cần **kế thừa class DBConnect** là có thể sử dụng **\_conn** thoải mái ko cần khởi tạo lại.

Các bạn nhớ sửa lại connection string cho chuẩn bên máy của các bạn nhé. Ở đây vì bài tập đơn giản nên ta chịu khó hard-code vậy :D. Chúng ta có nhiều cách khác nhau để tránh hard-code nhưng mình sẽ nói sau ở các bài khác.

Mình sẽ tạo file **DAL\_ThanhVien**.cs (Class file)

Ở đây mình sẽ làm sẵn luôn 4 methods là: **Lấy tất cả, Thêm, Xóa, Sửa** nhé

**DAL\_ThanhVien**.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data;

using System.Data.SqlClient;

using DTO\_QuanLy;

namespace DAL\_QuanLy

{

public class DAL\_ThanhVien : DBConnect

{

/// <summary>

/// Get toàn bộ thành viên

/// </summary>

/// <returns></returns>

public DataTable getThanhVien()

{

SqlDataAdapter da = new SqlDataAdapter("SELECT \* FROM THANHVIEN", \_conn);

DataTable dtThanhvien = new DataTable();

da.Fill(dtThanhvien);

return dtThanhvien;

}

/// <summary>

/// Thêm thành viên

/// </summary>

/// <param name="tv"></param>

/// <returns></returns>

public bool themThanhVien(DTO\_ThanhVien tv)

{

try

{

// Ket noi

\_conn.Open();

// Query string - vì mình để TV\_ID là identity (giá trị tự tăng dần) nên ko cần fải insert ID

string SQL = string.Format("INSERT INTO THANHVIEN(TV\_NAME, TV\_PHONE, TV\_EMAIL) VALUES ('{0}', '{1}', '{2}')", tv.THANHVIEN\_NAME, tv.THANHVIEN\_PHONE, tv.THANHVIEN\_EMAIL);

// Command (mặc định command type = text nên chúng ta khỏi fải làm gì nhiều).

SqlCommand cmd = new SqlCommand(SQL, \_conn);

// Query và kiểm tra

if (cmd.ExecuteNonQuery() > 0)

return true;

}

catch (Exception e)

{

}

finally

{

// Dong ket noi

\_conn.Close();

}

return false;

}

/// <summary>

/// Sửa thành viên

/// </summary>

/// <param name="tv"></param>

/// <returns></returns>

public bool suaThanhVien(DTO\_ThanhVien tv)

{

try

{

// Ket noi

\_conn.Open();

// Query string

string SQL = string.Format("UPDATE THANHVIEN SET TV\_NAME = '{0}', TV\_PHONE = '{1}', TV\_EMAIL = '{2}' WHERE TV\_ID = {3}", tv.THANHVIEN\_NAME, tv.THANHVIEN\_PHONE, tv.THANHVIEN\_EMAIL, tv.THANHVIEN\_ID);

// Command (mặc định command type = text nên chúng ta khỏi fải làm gì nhiều).

SqlCommand cmd = new SqlCommand(SQL, \_conn);

// Query và kiểm tra

if (cmd.ExecuteNonQuery() > 0)

return true;

}

catch (Exception e)

{

}

finally

{

// Dong ket noi

\_conn.Close();

}

return false;

}

/// <summary>

/// Xóa thành viên

/// </summary>

/// <param name="tv"></param>

/// <returns></returns>

public bool xoaThanhVien(int TV\_ID)

{

try

{

// Ket noi

\_conn.Open();

// Query string - vì xóa chỉ cần ID nên chúng ta ko cần 1 DTO, ID là đủ

string SQL = string.Format("DELETE FROM THANHVIEN WHERE TV\_ID = {0})", TV\_ID);

// Command (mặc định command type = text nên chúng ta khỏi fải làm gì nhiều).

SqlCommand cmd = new SqlCommand(SQL, \_conn);

// Query và kiểm tra

if (cmd.ExecuteNonQuery() > 0)

return true;

}

catch (Exception e)

{

}

finally

{

// Dong ket noi

\_conn.Close();

}

return false;

}

}

}

### **Xây dựng mô hình 3 lớp: Xây dựng Business**

Bước này là bước xử lý business logic (Business layer).

Ở bước này, ta có thể lấy dữ liệu từ **DAL** về, xử lý ABC XYZ gì đó rồi trả về lại cho **GUI** sử dụng. Hoặc khi update dữ liệu trên DB, **GUI** gửi data lên **BUS** và rồi ta xử lý ABC XYZ gì đó cho data của chúng ta, rồi mới insert/update/delete chẳng hạn,…

Vì app mình build là **app đơn giản**, nên mình chỉ cần gọi lên DAL và trả về tương ứng cho GUI xài thui

Mình sẽ tạo **BUS\_ThanhVien**.cs (Class file):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data;

using DAL\_QuanLy;

using DTO\_QuanLy;

namespace BUS\_QuanLy

{

public class BUS\_ThanhVien

{

DAL\_ThanhVien dalThanhVien = new DAL\_ThanhVien();

public DataTable getThanhVien()

{

return dalThanhVien.getThanhVien();

}

public bool themThanhVien(DTO\_ThanhVien tv)

{

return dalThanhVien.themThanhVien(tv);

}

public bool suaThanhVien(DTO\_ThanhVien tv)

{

return dalThanhVien.suaThanhVien(tv);

}

public bool xoaThanhVien(int TV\_ID)

{

return dalThanhVien.xoaThanhVien(TV\_ID);

}

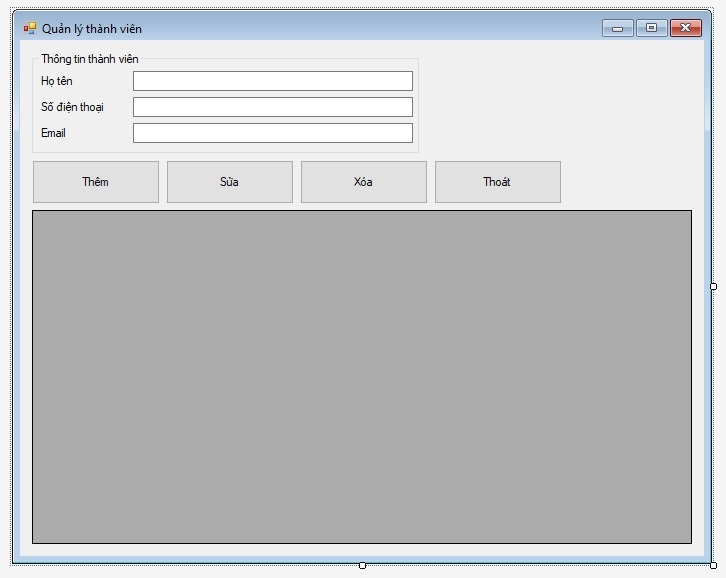
}

}

Chỉ đơn giản là gọi hàm và trả về thôi (app đơn giản mà @@)

### **Xây dựng mô hình 3 lớp: Xây dựng GUI**

Về GUI thì mình sẽ design như sau (mình sẽ gửi source đầy đủ ở trang cuối nhé):



Và mình sẽ gán lần lượt các chức năng vào nhé:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using DTO\_QuanLy;

using BUS\_QuanLy;

namespace GUI\_QuanLy

{

public partial class GUI\_ThanhVien : Form

{

BUS\_ThanhVien busTV = new BUS\_ThanhVien();

public GUI\_ThanhVien()

{

InitializeComponent();

}

private void btnExit\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void btnAdd\_Click(object sender, EventArgs e)

{

if (txtEmail.Text != "" && txtName.Text != "" && txtSDT.Text != "")

{

// Tạo DTo

DTO\_ThanhVien tv = new DTO\_ThanhVien(0, txtName.Text, txtSDT.Text, txtEmail.Text); // Vì ID tự tăng nên để ID số gì cũng dc

// Them

if (busTV.themThanhVien(tv))

{

MessageBox.Show("Thêm thành công");

dgvTV.DataSource = busTV.getThanhVien(); // refresh datagridview

}

else

{

MessageBox.Show("Thêm ko thành công");

}

}

else

{

MessageBox.Show("Xin hãy nhập đầy đủ");

}

}

private void GUI\_ThanhVien\_Load(object sender, EventArgs e)

{

dgvTV.DataSource = busTV.getThanhVien(); // get thanh vien

}

private void btnEdit\_Click(object sender, EventArgs e)

{

// Kiểm tra nếu có chọn table rồi

if (dgvTV.SelectedRows.Count > 0)

{

if (txtEmail.Text != "" && txtName.Text != "" && txtSDT.Text != "")

{

// Lấy row hiện tại

DataGridViewRow row = dgvTV.SelectedRows[0];

int ID = Convert.ToInt16(row.Cells[0].Value.ToString());

// Tạo DTo

DTO\_ThanhVien tv = new DTO\_ThanhVien(ID, txtName.Text, txtSDT.Text, txtEmail.Text); // Vì ID tự tăng nên để ID số gì cũng dc

// Sửa

if (busTV.suaThanhVien(tv))

{

MessageBox.Show("Sửa thành công");

dgvTV.DataSource = busTV.getThanhVien(); // refresh datagridview

}

else

{

MessageBox.Show("Sửa ko thành công");

}

}

else

{

MessageBox.Show("Xin hãy nhập đầy đủ");

}

}

else

{

MessageBox.Show("Hãy chọn thành viên muốn sửa");

}

}

private void dgvTV\_Click(object sender, EventArgs e)

{

// Lấy row hiện tại

DataGridViewRow row = dgvTV.SelectedRows[0];

// Chuyển giá trị lên form

txtName.Text = row.Cells[1].Value.ToString();

txtSDT.Text = row.Cells[2].Value.ToString();

txtEmail.Text = row.Cells[3].Value.ToString();

}

private void btnDelete\_Click(object sender, EventArgs e)

{

// Kiểm tra nếu có chọn table rồi

if (dgvTV.SelectedRows.Count > 0)

{

// Lấy row hiện tại

DataGridViewRow row = dgvTV.SelectedRows[0];

int ID = Convert.ToInt16(row.Cells[0].Value.ToString());

// Xóa

if (busTV.xoaThanhVien(ID))

{

MessageBox.Show("Xóa thành công");

dgvTV.DataSource = busTV.getThanhVien(); // refresh datagridview

}

else

{

MessageBox.Show("Xóa ko thành công");

}

}

else

{

MessageBox.Show("Hãy chọn thành viên muốn xóa");

}

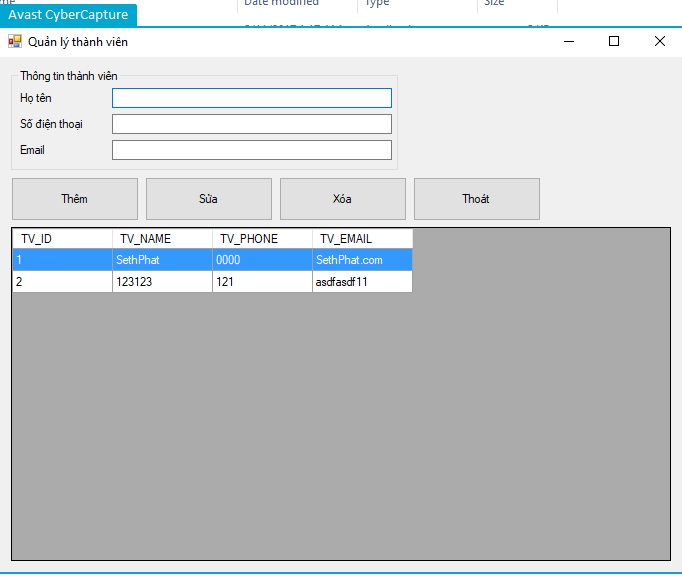
}

}

}

### **Xây dựng mô hình 3 lớp:**

Sau khi viết xong thì toàn bộ chương trình demo của mình.



Mô hình 3 lớp có lợi và hại hết nhưng hầu như lợi nhiều hơn.

# CHƯƠNG 4: TÀI LIỆU THAM KHẢO

* 1. Công nghệ ADO.NET:

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/>

* 1. Hiển thị dữ liệu lên datagridview:

<https://www.youtube.com/watch?v=fQYAJr57iQM&ab_channel=Kteam>

* 1. Mô hình 3 lớp:

<https://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET>

<https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648105(v=pandp.10)>