

TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN
--oOo--

THÁI DUY QUÝ
PHAN THỊ THANH NGA
TRẦN THỊ PHƯƠNG LINH

GIÁO TRÌNH
LẬP TRÌNH CƠ SỞ DỮ LIỆU

Đà Lạt, Tháng 8 năm 2020

LỜI MỞ ĐẦU

Sự phát triển như vũ bão của cuộc Cách mạng 4.0 đã kéo theo sự thay đổi lớn trong nhận thức của nhiều người. Với sự phát triển nhanh chóng của khoa học - kỹ thuật - công nghệ, các ứng dụng về chuyển đổi số cũng đã trở nên phong phú, đa dạng và đóng vai trò không thể thiếu trong sự phát triển chung của nền kinh tế. Những chương trình ứng dụng, những phần mềm quản lý được thay thế cho việc quản lý bằng hồ sơ, sổ sách đã quá lạc hậu, tốn chi phí và thời gian. Từ những ứng dụng quản lý trên máy tính, đến website tin tức, thương mại, đến các ứng dụng trên các thiết bị điện tử thông minh, lớn hơn nữa là những chương trình quản lý hoạt động của cả một hệ thống điều khiển tự động, tất cả đều là thành tựu vượt bậc của công nghệ thông tin.

Với sự phát triển không ngừng đó, các ứng dụng mới ngày càng đòi hỏi những tính năng ưu việt hơn, thông minh hơn. Không những về tính linh hoạt, thời gian xử lý mà còn phải kể đến khả năng lưu trữ lượng dữ liệu cực kỳ lớn với tốc độ truy xuất, thực thi lệnh nhanh chóng và hiệu quả. Để đáp ứng điều này, ngoài việc sử dụng hệ quản trị cơ sở dữ liệu phù hợp, chúng ta cũng cần phải có các công cụ, môi trường tốt để phát triển ứng dụng. Quan trọng hơn, cần phải biết cách tổ chức chương trình một cách khoa học và tinh tế nhằm cải thiện hiệu suất và nâng cao chất lượng của phần mềm ứng dụng. Đồng thời, cũng phải biết áp dụng các thành tựu nghiên cứu, các công cụ hiệu quả và mới nhất hiện nay vào trong các ứng dụng.

Nhằm nâng cao kiến thức cho sinh viên và những người quan tâm nghiên cứu ngành Công nghệ Thông tin, lĩnh vực lập trình, chúng tôi biên soạn và giới thiệu “Giáo trình Lập trình Cơ sở dữ liệu” này tới các độc giả như một tài liệu học tập, nghiên cứu. Giáo trình cung cấp một lượng kiến thức xuyên suốt từ khâu tìm hiểu về cơ chế lập trình Windows đến việc kết nối cơ sở dữ liệu trên Hệ quản trị Cơ sở dữ liệu SQL Server. Đây là cuốn giáo trình kết hợp giữa việc giới thiệu lập trình trên Windows và việc kết nối cơ sở dữ liệu. Thông qua cuốn giáo trình, độc giả vừa tìm hiểu kiến thức phát triển ứng dụng, vừa tích luỹ cho bản thân những kỹ năng cần thiết trong việc xây dựng các phần mềm ứng dụng. Lượng kiến thức là vô hạn, vì thế, trong giáo trình này, chúng tôi không thể đề cập được hết những khía cạnh của việc phát triển phần mềm. Thay vào đó, chúng tôi hi vọng, đây sẽ là tài liệu tham khảo tương đối đầy đủ về lập trình ứng dụng với cơ sở dữ liệu để quý bạn đọc tự khám phá và phát huy những kỹ năng tiềm ẩn của mình.

Giáo trình được chia làm bảy chương với nội dung như sau:

Chương 1 giới thiệu và trình bày môi trường phát triển Visual Studio (ngôn ngữ C#), một số công cụ để thiết kế và lập trình chương trình bằng Windows Form. Ngoài ra, chương này cũng hướng dẫn cách sử dụng các thuộc tính, các phương thức và sự kiện cơ bản để thao tác và lập trình các ứng dụng trên ngôn ngữ C#.

Chương 2 trình bày tổng quan về một số kiểu cơ sở dữ liệu bán cấu trúc để lưu trữ dữ liệu. Chương này minh họa một số kiểu cơ sở dữ liệu bán cấu trúc như XML, JSON, BSON với ngôn ngữ lập trình C#.

Chương 3 hướng dẫn chi tiết cách thiết lập một kết nối đến cơ sở dữ liệu. Mọi thao tác xử lý có liên quan đến cơ sở dữ liệu phải thông qua một phương tiện liên lạc được gọi là đối tượng kết nối (Connection).

Chương 4 đi sâu phân tích các bước để thực thi các lệnh truy vấn cơ sở dữ liệu như SELECT, INSERT, UPDATE và DELETE thông qua đối tượng Command. Ngoài ra, chương này cũng nêu các phương pháp sử dụng đối tượng Command để gọi một thủ tục, thực thi thủ tục theo các tham số và lấy về các dòng, các cột trong một bảng hay một khung nhìn cụ thể.

Chương 5 giới thiệu một số thư viện hỗ trợ việc đọc và ghi tập tin dạng văn bản, tập tin nhị phân, các tập tin bán cấu trúc như CSV, XML và các bảng tính Excel. Chương này cũng đi sâu tìm hiểu thư viện Entities Framework và cách sử dụng thư viện đó trong chương trình ứng dụng.

Chương 6 trình bày cách điều khiển các giao dịch nâng cao sử dụng SQL Server và ADO .Net nhằm nâng cao hiệu suất của sản phẩm phần mềm. Chương này cũng đưa ra các ví dụ minh họa các giao dịch trên SQL Server và trên ADO .Net.

Chương 7 giới thiệu về mô hình đa tầng (n-tier), kiến trúc của mô hình đa tầng và minh họa cách xây dựng ứng dụng đơn giản với mô hình đa tầng từ khâu xây dựng cơ sở dữ liệu cho tới phần ứng dụng.

Chúng tôi hi vọng giáo trình này sẽ là một tài liệu tham khảo có ích đối với sinh viên ngành công nghệ thông tin và những độc giả quan tâm tới lĩnh vực xây dựng ứng dụng kết nối cơ sở dữ liệu. Chúng tôi rất mong nhận được sự cổ vũ và những ý kiến đóng góp quý báu của các quý độc giả giúp bổ sung và hoàn thiện hơn nữa giáo trình.

Cuối cùng, chúng tôi xin gửi lời cảm ơn đến các thầy cô và đồng nghiệp đã ủng hộ và giúp đỡ chúng tôi hoàn thành giáo trình này.

Dà Lạt, tháng 8 năm 2020

Các tác giả

Thái Duy Quý,
Phan Thị Thanh Nga,
Trần Thị Phương Linh

MỤC LỤC

LỜI MỞ ĐẦU	3
MỤC LỤC	5
CHƯƠNG 1. TỔNG QUAN VỀ LẬP TRÌNH GIAO DIỆN.....	9
1.1. Giới thiệu môi trường .NET và Windows Form	9
1.1.1. Cách tạo Windows Form	10
1.1.2. Các thuộc tính của Form	12
1.1.3. Các sự kiện của Form.....	14
1.1.4. Các phương thức của Form.....	15
1.1.5. Các loại Form.....	17
1.2. Các điều khiển cơ bản.....	18
1.2.1. Điều khiển Label.....	19
1.2.2. Điều khiển TextBox.....	20
1.2.3. Điều khiển Button	22
1.2.4. Điều khiển ComboBox và điều khiển ListBox	24
1.2.5. Điều khiển CheckBox và điều khiển RadioButton	26
1.3. Điều khiển chứa các điều khiển khác	27
1.3.1. GroupBox.....	27
1.3.2. Panel.....	27
1.3.3. TabControl	28
1.4. Một số điều khiển đặc biệt.....	30
1.4.1. MaskedTextBox	30
1.4.2. RichTextBox	31
1.4.3 NumericUpDown và DomainUpDown.....	33
1.4.4. MonthCalendar và DateTimePicker	34
1.4.5. Timer và ProgressBar	36
1.4.6. PictureBox và ImageList.....	38
1.4.7. Lớp MessageBox	41
1.5. Một số điều khiển nâng cao	43
1.5.1. Điều khiển ListView	43
1.5.2. Điều khiển TreeView	46
1.5.3. Điều khiển ToolTip, ErrorProvider.....	49
1.5.4. Điều khiển UserControl	50
1.6. Các điều khiển kiểu Menu	51
1.6.1. ToolStrip	51
1.6.2. ContextMenuStrip	52
1.6.3. ToolStrip	53
1.6.4. StatusStrip	54
1.7. Các dạng hộp thoại (Dialog).....	54
1.7.1. OpenFileDialog và SaveFileDialog	54
1.7.2. ColorDialog và FontDialog	56
1.8. Xử lý sự kiện chuột, bàn phím.....	57
1.8.1. Xử lý sự kiện chuột	57
1.8.2. Xử lý sự kiện bàn phím.....	59
1.9. Kết chương	60
Bài tập:.....	60
CHƯƠNG 2. BIỂU DIỄN DỮ LIỆU BÁN CẤU TRÚC	62
2.1. Dữ liệu bán cấu trúc.....	62
2.2. Kiểu dữ liệu XML	64

2.2.1. Cú pháp của XML.....	64
2.2.2. Thuộc tính trong XML.....	65
2.2.3. Cách tạo tập tin XML bằng Visual Studio.....	65
2.2.4. Tạo chuỗi kết nối trong tập tin App.config.....	67
2.3. Kiểu dữ liệu JSON.....	67
2.3.1. Cú pháp của JSON.....	68
2.3.2. Các kiểu dữ liệu trong JSON	69
2.3.3. Cách tạo tập tin JSON bằng Visual Studio	70
2.3.4. Đọc tập tin JSON bằng Visual Studio với ngôn ngữ C#.....	71
2.4. Kiểu dữ liệu BSON.....	74
2.5. Kết chương	77
Bài tập:.....	77
CHƯƠNG 3. KẾT NỐI CƠ SỞ DỮ LIỆU	79
3.1. Tạo đối tượng kết nối	79
3.1.1. Kết nối tới cơ sở dữ liệu SQL Server.....	79
3.1.2. Kết nối cơ sở dữ liệu Access.....	82
3.1.3. Kết nối cơ sở dữ liệu Oracle	83
3.1.4. Kết nối cơ sở dữ liệu MySQL.....	84
3.1.5. Kết nối cơ sở dữ liệu SQLite	84
3.1.6. Kết nối tới file *.xlsx	85
3.2. Mở và đóng kết nối.....	85
3.3. Tổ hợp kết nối – <i>Connection Pooling</i>	86
3.4. Quản lý trạng thái kết nối	89
3.5. Quản lý sự kiện trong quá trình kết nối	89
3.5.1. Sự kiện StateChange	89
3.5.2. Sự kiện InfoMessage.....	91
3.6. Kết chương	93
Bài tập.....	93
CHƯƠNG 4. THỰC THI LỆNH THAO TÁC TRÊN CƠ SỞ DỮ LIỆU	94
4.1. Thao tác truy vấn cơ sở dữ liệu và đọc kết quả trả về	94
4.1.1. Lớp SqlCommand	94
4.1.2. Thực thi các lệnh SELECT và TableDirect	96
4.1.3. Thực thi các lệnh làm thay đổi thông tin trong cơ sở dữ liệu	98
4.1.4. Thực thi các lệnh INSERT, UPDATE và DELETE	98
4.1.5. Các giao dịch trong cơ sở dữ liệu	99
4.1.6. Truyền tham số vào các lệnh	100
4.1.7. Gọi các thủ tục SQL Server	103
4.2. Thao tác cơ sở dữ liệu với lớp SqlDataReader	107
4.2.1. Tạo đối tượng SqlDataReader.....	107
4.2.2. Đọc dữ liệu từ SqlDataReader	107
4.2.3. Lấy giá trị từ các cột với kiểu cụ thể.....	108
4.2.4. Sử dụng các phương thức dạng Get* để đọc dữ liệu	109
4.2.5. Thực thi nhiều lệnh truy vấn SQL	110
4.3. Lưu trữ dữ liệu với Dataset.....	111
4.3.1. Lớp SqlDataAdapter	111
4.3.2. Lớp DataSet	112
4.3.3. Lớp DataTable	114
4.4. Trích lọc và sắp xếp dữ liệu dùng DataView	114
4.5. Kết chương	117
Bài tập.....	117

CHƯƠNG 5. CÁC THƯ VIỆN HỖ TRỢ LẬP TRÌNH CSDL	119
5.1. Đọc và tạo tập tin dạng văn bản	119
5.2. Đọc và ghi tập tin định dạng nhị phân.....	122
5.3. Đọc và ghi nội dung từ tập tin XML	124
5.4. Đọc và tạo tập tin CSV	128
5.5. Đọc và tạo bảng tính Excel.....	130
5.6. Entity Framework 6	133
5.6.1. Định nghĩa.....	134
5.6.2. Các tính năng của Entity Framework.....	134
5.6.3. Các phiên bản.....	135
5.6.4. Luồng xử lý trong Entity Framework	136
5.6.5. Mô hình dữ liệu thực thể.....	137
5.6.6. Kiến trúc của Entity Framework.....	138
5.6.7. Lớp ngữ cảnh trong Entity Framework.....	139
5.6.8. Thực thể	140
5.6.9. Các loại thực thể trong Entity Framework.....	144
5.6.10. Trạng thái của thực thể.....	146
5.6.11. Các hướng tiếp cận đối với Entity Framework	147
5.6.12. Các tình huống lưu trữ trong Entity Framework.....	149
5.6.13. Lớp DbContext	150
5.6.14. Lớp DbSet	153
5.6.15. Mối quan hệ giữa các thực thể trong EF6.....	155
5.7. Lưu trữ dữ liệu trong tình huống có kết nối	158
5.7.1. Thêm dữ liệu mới.....	159
5.7.2. Cập nhật dữ liệu	160
5.7.3. Xóa dữ liệu.....	161
5.7.4. Truy vấn dữ liệu	161
5.7.5. Eager loading	167
5.7.6. Lazy loading.....	170
5.7.7. Explicit Loading.....	171
5.7.8. Thực thi một truy vấn thô SQL.....	172
5.7.9. Lưu trữ dữ liệu trong tình huống không kết nối	174
5.8. Kết chương	177
Bài tập:.....	178
CHƯƠNG 6. KIỂM SOÁT GIAO DỊCH NÂNG CAO	180
6.1. Các lớp SqlTransaction.....	180
6.2. Thiết lập điểm lưu (<i>Save point</i>)	182
6.2.1. Thiết lập điểm lưu bằng lệnh trong SQL Server	182
6.2.2. Tạo điểm lưu bằng đối tượng SqlTransaction.....	183
6.3. Mức độ độc lập giao dịch cơ sở dữ liệu.....	186
6.3.1. Phantoms	186
6.3.2. Nonrepeatable reads	187
6.3.3. Dirty reads	188
6.3.4. Thiết lập mức độ độc lập giao dịch trong SQL Server	189
6.3.5. Gán mức độ độc lập giao dịch trong ADO .Net.....	190
6.4. Các chế độ khóa của SQL Server	192
6.4.1. Các loại khóa.....	192
6.4.2. Các chế độ khóa	192
6.5. Các phương pháp chặn giao dịch.....	193
6.5.1. Chặn giao dịch	193

6.5.2. Đặt thời gian hủy bỏ cho khóa	194
6.5.3. Chặn và đọc các giao dịch kiểu Serializable/Repeatable trong ADO .Net	194
6.5.4. Deadlocks.....	196
6.6. Kết chương	199
Bài tập.....	199
CHƯƠNG 7. MÔ HÌNH ĐA TẦNG.....	200
7.1. Giới thiệu mô hình đa tầng	200
7.2. Tạo ứng dụng mô hình đa tầng trên Visual Studio.....	202
7.3. Tầng truy cập dữ liệu (<i>DataAccess</i>)	206
7.3.1. Xây dựng cơ sở dữ liệu	206
7.3.2. Xây dựng lớp các tham số chung.....	208
7.3.3. Xây dựng các lớp ánh xạ bảng.....	209
7.3.4. Xây dựng các lớp truy xuất dữ liệu.....	210
7.4. Tầng xử lý logic (<i>BusinessLogic - BLL</i>)	213
7.4.1. Lớp CategoryBL	213
7.4.2. Lớp FoodBL.....	214
7.5. Lớp giao diện người dùng (User Interface – UI)	215
7.5.1. Tạo chuỗi kết nối	215
7.5.2. Thiết kế giao diện chương trình	216
7.5.3. Các phương thức tự động tải dữ liệu.....	218
7.5.4. Xử lý nút thêm, xoá, sửa	221
7.6. Kết chương	224
Bài tập:.....	224
TÀI LIỆU THAM KHẢO	225
Phụ lục:.....	226

CHƯƠNG 1. TỔNG QUAN VỀ LẬP TRÌNH GIAO DIỆN

Ngày nay, khi làm ra một sản phẩm phần mềm, yếu tố góp phần không nhỏ vào sự thành công của sản phẩm là giao diện người dùng đồ họa (*Graphic User Interface - GUI*), thường gọi tắt là Giao diện người dùng, Giao diện, hay đơn giản là *GUI*. Giao diện trong sản phẩm phần mềm như một bảng chỉ dẫn không lời, thực hiện sứ mệnh mang lại sự tiện lợi, đơn giản, và hiệu quả cho người dùng. Một giao diện có sự bố trí các yếu tố nội dung và hình thức kết hợp nhau một cách mạch lạc, màu sắc hài hòa, tính năng thân thiện, hiển thị nhanh và dễ sử dụng không những đáp ứng được tính thẩm mỹ mà còn thỏa mãn được các yêu cầu đặc thù của người dùng. Bên cạnh đó, *GUI* giúp người dùng nhanh chóng nắm rõ cách sử dụng, thích ứng và giảm thời gian làm quen với ứng dụng mới. Nó cho phép người dùng tương tác với hệ thống thông qua các nhẫn, nút bấm, ô nhập liệu, v.v. bằng cách sử dụng chuột, con trỏ hay bàn phím.

Một trong những công cụ phổ biến hỗ trợ lập trình viên tạo ra các giao diện người dùng một cách hiệu quả đó là Windows Form của Microsoft trong môi trường .NET. Windows Form cho phép các lập trình viên tạo các Form một cách dễ dàng, hỗ trợ nhiều điều khiển (*controls*) và các thành phần (*components*) giúp cho lập trình viên có thể xây dựng và tùy chỉnh giao diện một cách nhanh chóng và đa dạng, phù hợp nhu cầu và mục đích của người sử dụng.

Chương này giới thiệu những vấn đề cơ bản trong lập trình giao diện: Cách tạo Form với ngôn ngữ C#; Một số trình điều khiển cơ bản và nâng cao; Các hộp thoại; Xử lý các sự kiện chuột và bàn phím trong Form... Các vấn đề được đề cập trong chương bao gồm:

- Giới thiệu về Windows Form;
- Các điều khiển cơ bản;
- Các điều khiển nâng cao;
- Xử lý sự kiện chuột, bàn phím.

1.1. Giới thiệu môi trường .NET và Windows Form

.NET Framework là một nền tảng lập trình của Microsoft, cung cấp cho lập trình viên các thư viện dùng chung hỗ trợ việc phát triển các kiểu ứng dụng khác nhau như: ASP.NET, Windows Form, Web Services, Windows Services... .NET Framework gồm hai thành phần chính đó là Common Language Runtime (*CLR*) và Thư viện lớp:

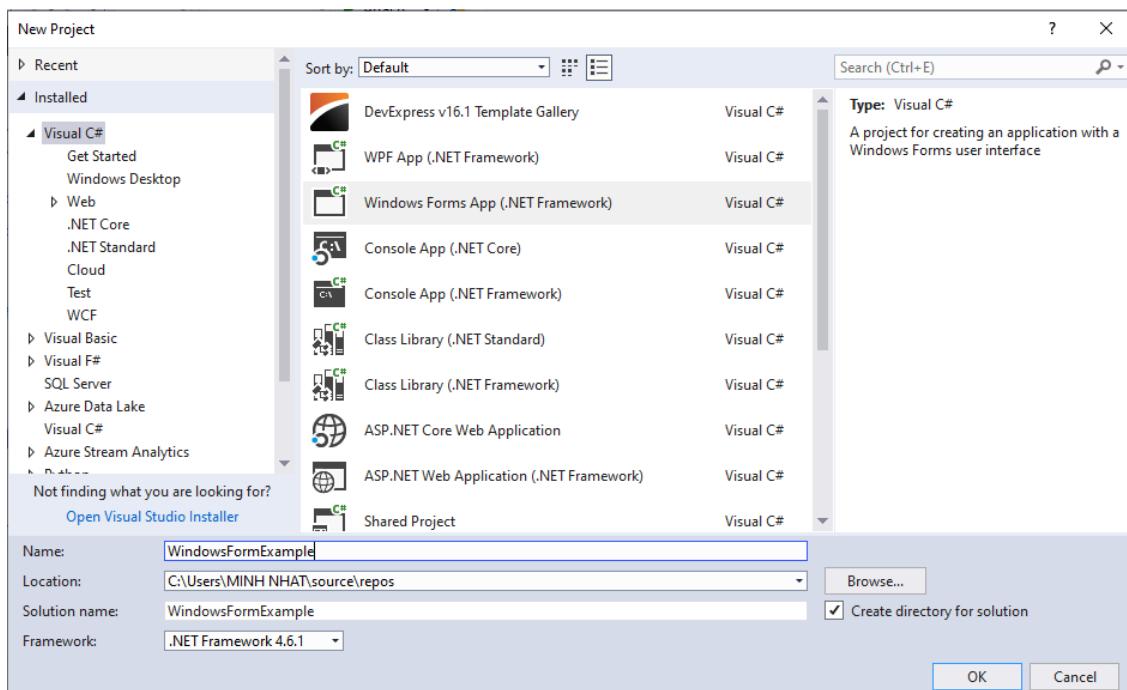
- *CLR*: Là một nền tảng của .NET Framework, giúp tích hợp nhiều ngôn ngữ lập trình khác nhau, như: ASP.NET, C#.NET, VB.NET,... vào bộ công cụ lập trình Visual Studio .NET. Đồng thời giúp các ứng dụng viết trên các ngôn ngữ này có thể chạy được chung trên nền tảng hệ điều hành Windows. CLR sử dụng một trình biên dịch gọi là Just-In-Time (JIT) để chuyển các đoạn mã trung gian thành mã máy rồi sau đó tiến hành thực thi;

• **Thư viện lớp:** Là một tập hợp các lớp được viết ra dưới dạng các thư viện (*.dll). Những lớp này được xây dựng một cách trực quan và dễ sử dụng, cho phép các lập trình viên thao tác rất nhiều các tác vụ trên Windows. Thư viện lớp trong .NET Framework được tổ chức thành các nhóm, gọi là các không gian tên (*namespace*). Mỗi không gian tên bao gồm các lớp được sử dụng cho một chức năng cụ thể. Trong chương này chúng ta quan tâm đến bộ thư viện Windows Form, chứa trong không gian tên System.Windows.Forms bao gồm các lớp đối tượng được sử dụng để phục vụ cho việc xây dựng các ứng dụng Windows cơ bản; Một số thư viện con thường dùng trong bộ này đó là: Form, Control, Panel, Button, TextBox, UserControl,...

Windows Form chính là giao diện người dùng đồ họa (GUI) cho phép người dùng tương tác trực quan với hệ thống hay một chương trình phần mềm cụ thể. Tất cả những vấn đề liên quan đến việc tạo, thiết kế, sử dụng Form sẽ được trình bày bởi các mục sau đây.

1.1.1. Cách tạo Windows Form

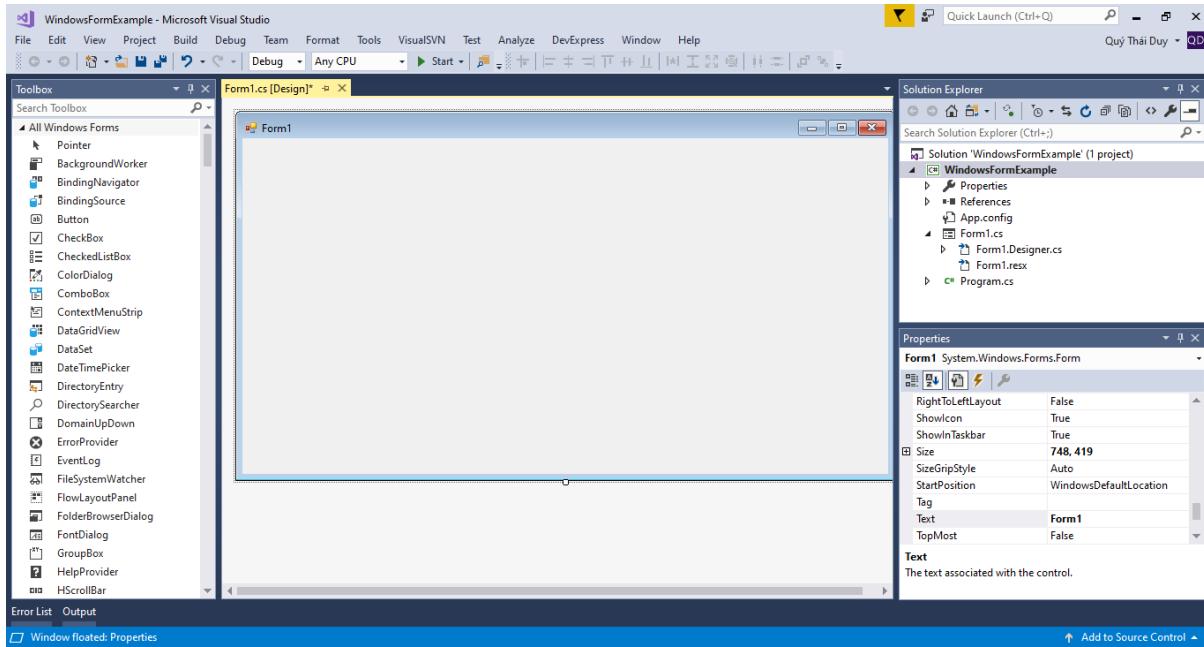
Để tạo một dự án Windows Form với ngôn ngữ C#, sử dụng Visual Studio, ta làm như sau: Khởi động Visual Studio; Vào File chọn New chọn Project; Cửa sổ New Project xuất hiện (Hình 1.1) cho phép chọn ngôn ngữ (ở đây là C#), loại ứng dụng muốn viết (Windows Form), đặt tên cho dự án trong cửa sổ Name, chọn thư mục chứa dự án và nhấn OK. Ở đây, có thể đánh dấu vào hộp kiểm “Create directory for solution” để tạo một thư mục gốc cho dự án.



Hình 1.1. Giao diện tạo dự án Windows Forms bằng ngôn ngữ C#

Khi tạo xong, dự án sẽ xuất hiện một cửa sổ mặc định (Hình 1.2), có tên do người dùng tạo (sau đây gọi là Form). Lúc này lập trình viên có thể đổi tên, thiết kế giao diện

của cửa sổ bằng cách thêm các điều khiển hay các thành phần cần thiết phù hợp với các chức năng của phần mềm.



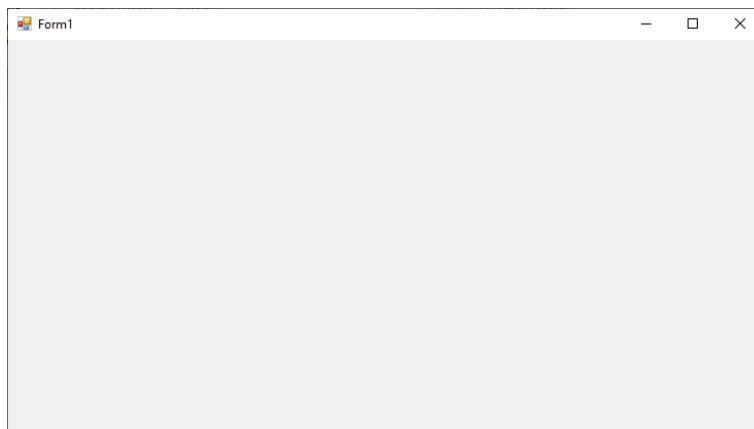
Hình 1.2. Kết quả tạo dự án Windows Forms

Khi Form được tạo xong, Visual Studio có thiết lập mặc định một số cửa sổ để người dùng thao tác, các cửa sổ bao gồm:

- *Toolbox*: Là cửa sổ chứa các trình điều khiển (controls) như nút bấm, hộp kiểm, ô nhập.... Cửa sổ Toolbox thường nằm bên trái màn hình, người dùng có thể bật cửa sổ này bằng cách vào View chọn Toolbox hoặc nhấn tổ hợp phím Ctrl + Alt + X;
- *Solution Explorer*: Là cửa sổ chứa các tập tin của dự án. Nhìn cửa sổ Explorer trong Hình 1.2 chúng ta thấy ngoài Form do người dùng tạo ra còn có các tập tin khác được hệ thống tự thêm vào như: Các tập tin thư viện trong References; Tập tin App.config chứa thông tin cấu hình; Tập tin Program.cs chứa các lệnh dùng để khởi tạo chương trình. Ngoài ra, khi nhìn lên phía trên, chúng ta thấy một thành phần được gọi là Solution, ở đây Solution gọi là giải pháp, một giải pháp chứa nhiều dự án (*project*). Cửa sổ Solution Explorer thường nằm bên phải màn hình, người dùng có thể bật lên bằng cách vào View, chọn Solution Explorer hoặc nhấn tổ hợp phím Ctrl + Alt + L;
- *Properties*: Là cửa sổ chứa các thuộc tính (*properties*) và sự kiện của Form hoặc điều khiển. Mỗi khi chọn một đối tượng, hộp thoại này sẽ tự động hiện ra các thuộc tính liên quan của đối tượng đó. Cửa sổ này cũng thường nằm bên phải màn hình, người dùng có thể bật lên bằng cách click phải lên đối tượng, chọn Properties hoặc nhấn tổ hợp phím Alt + Enter;
- *Error List*: Là hộp thoại chứa các lỗi sai khi biên dịch, hộp thoại này thường tự động hiển thị khi có lỗi và nằm ở phía dưới màn hình.

Có hai giao diện dùng để thao tác chương trình là: Design (thiết kế) và Code (mã nguồn). Giao diện Design dùng để thao tác bằng cách kéo thả trực quan, trong khi đó giao diện Code dùng để thực hiện viết mã lệnh chương trình. Tùy vào tình huống xử lý mà lập trình viên thực hiện trên giao diện nào. Muốn vào giao diện Design, chỉ cần nhấp chuột lên Form.cs, muốn viết mã lệnh, có thể click phải lên Form, chọn ViewCode hoặc nhấp đúp chuột lên một điều khiển bất kỳ, hệ thống sẽ tự tạo sự kiện trong màn hình Code.

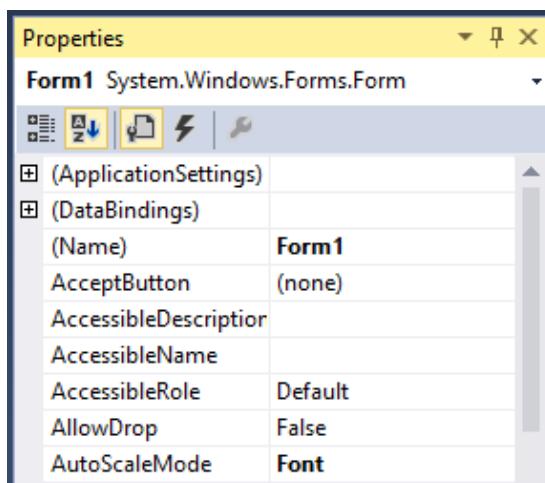
Để chạy chương trình, nhấn phím F5 hoặc nhấn vào nút Start trên thanh công cụ, nếu có lỗi cú pháp, hệ thống sẽ báo trong cửa sổ Error List, nếu không, chương trình sẽ chạy lên với một giao diện trống (Hình 1.3).



Hình 1.3. Giao diện khi chạy lần đầu tiên

1.1.2. Các thuộc tính của Form

Windows Form hỗ trợ việc tùy chỉnh giao diện của Form dựa trên việc thay đổi giá trị các thuộc tính được hiển thị trong cửa sổ Properties (Hình 1.4). Tùy vào mục đích sử dụng mà người dùng có thể thay đổi giá trị của các thuộc tính này. Bảng 1.1 là danh sách một số thuộc tính cơ bản của Form. Ngoài các thuộc tính riêng cho Form, còn có một số thuộc tính có thể dùng cho các trình điều khiển khác.

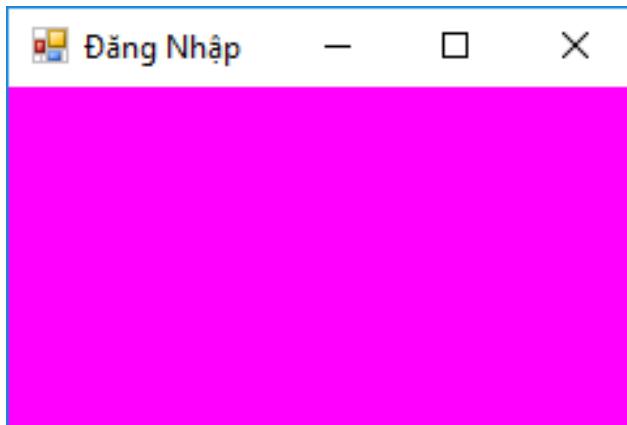


Hình 1.4. Cửa sổ Properties chứa các thuộc tính và sự kiện của Form

Bảng 1.1. Một số thuộc tính cơ bản của Form

Thuộc tính	Mô tả
Name	Đặt tên của Form, thường bắt đầu bởi tiền tố frm, ví dụ: frmLogin
AcceptButton	Thuộc tính này nhận giá trị là tên của một nút có trên Form. Khi đó, người dùng có thể nhấn phím Enter trên bàn phím thay vì click vào nút để thực thi
BackColor	Thiết lập màu nền cho Form
BackgroundImage	Thiết lập hình nền cho Form
CancelButton	Thuộc tính này nhận giá trị là tên của một nút chứa trên Form. Khi đó, người dùng có thể nhấn phím ESC trên bàn phím thay vì click vào nút để thực thi
Enable	Nhận giá trị true hoặc false, nếu thiết lập là false thì sẽ không cho phép người dùng thao tác trên các điều khiển chứa trong Form
Font	Các thiết lập cho văn bản của các điều khiển chứa trong Form như Font chữ, kích thước, kiểu...
ForeColor	Thiết lập màu mặc định cho văn bản của các điều khiển chứa trong Form
FormBorderStyle	Thiết lập kiểu đường biên của Form khi chạy chương trình
Icon	Biểu tượng hiển thị phía ngoài cùng bên trái trên thanh tiêu đề của Form
Location	Thiết lập vị trí hiển thị của Form trên màn hình (chỉ khi thuộc tính StartPosition được thiết lập là Manual)
MaximizeBox	Nhận giá trị true hoặc false, nếu thiết lập là false thì nút maximize của Form trên thanh titlebar sẽ không hiển thị
MaximumSize	Thiết lập kích thước tối đa của Form (chiều rộng và chiều cao)
MinimizeBox	Nhận giá trị true hoặc false, nếu thiết lập là false thì nút minimize của Form trên thanh titlebar sẽ không hiển thị
MinimumSize	Thiết lập kích thước tối thiểu của Form (rộng x cao)
Opacity	Thiết lập mức độ trong suốt của Form
Size	Thiết lập kích thước của Form (chiều rộng và chiều cao)
StartPosition	Vị trí hiển thị của Form trên màn hình
Text	Văn bản hiển thị trên thanh tiêu đề của Form
TopMost	Nhận giá trị true hoặc false, nếu thiết lập là true thì Form sẽ luôn hiển thị phía trên các cửa sổ khác
Visible	Nhận giá trị true hoặc false, nếu thiết lập là false thì Form sẽ không hiển thị, và ngược lại.
WindowState	Cách hiển thị Form khi khởi chạy, có ba giá trị đó là: <i>Normal</i> (bình thường), <i>Minimized</i> (thu nhỏ), <i>Maximized</i> (tối ưu)
IsMDIContainer	Nhận giá trị true hoặc false, nếu thiết lập là false thì Form ở dạng bình thường, nếu là true thì Form là MDI Form (<i>Multiple Document Interface</i>) – là kiểu Form cha, có thể có nhiều Form con trong nó.
MdiParent	Nhận giá trị là một đối tượng MDI Form. Khi thiết lập giá trị cho thuộc tính này thì Form hiện thời sẽ trở thành Child Form (Form con)

Ví dụ: Khi thiết lập thuộc tính Name = “frmLogin”, Text = “Đăng Nhập”, Size = “255,165”, và BackColor = “Fuchsia” thì khi chạy chương trình, ta được kết quả như trong Hình 1.5.

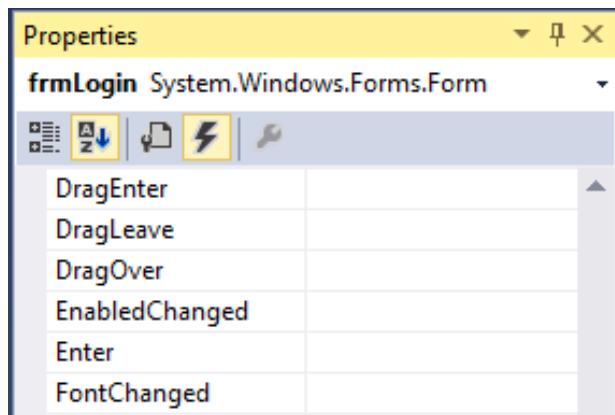


Hình 1.5. Ví dụ về thay đổi một số thuộc tính của Form

Lưu ý: Tiêu đề của Form thể hiện thông qua thuộc tính Text; Còn thuộc tính Name là tên của Form, dùng để thao tác khi sử dụng mã nguồn. Để dễ thao tác, các điều khiển (hoặc Form) khi đặt tên thường người ta hay cho thêm tiền tố phía trước. Ví dụ Form là frm..., Textbox là txt..., Button là btn...

1.1.3. Các sự kiện của Form

Các sự kiện có thể hiểu là các hành động bất ngờ liên quan đến sự vận hành của Form, ví dụ như mở Form, đóng Form, di chuột vào Form... Tất cả các sự kiện của Form hoặc một điều khiển được liệt kê trong cửa sổ Properties, click vào biểu tượng tia chớp để xem danh sách các sự kiện của Form (Hình 1.6).



Hình 1.6. Các sự kiện của Form trong cửa sổ Properties

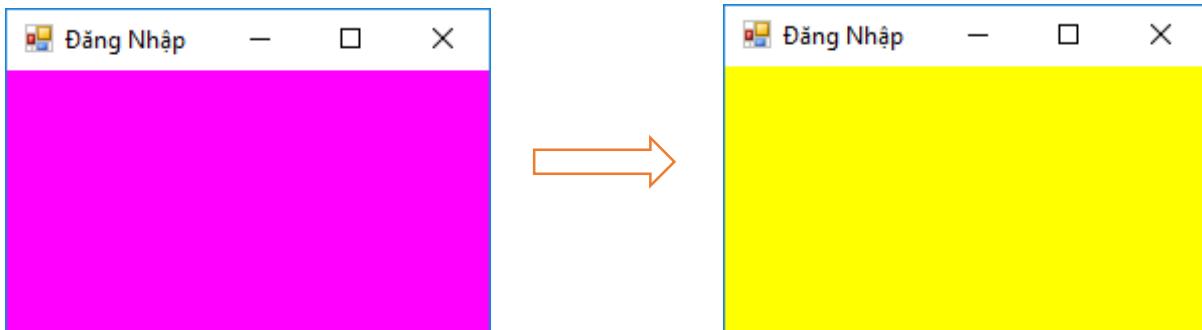
Muốn sử dụng sự kiện nào, chỉ cần click đúp chuột vào sự kiện đó. Khi đó, chương trình sẽ tự động phát sinh một đoạn mã lệnh cho sự kiện tương ứng. Chúng ta sẽ lập trình hành vi mong muốn cho sự kiện trong đoạn mã này. Bảng 1.2 thống kê một số sự kiện của Form, giống như thuộc tính, một số sự kiện có thể dùng chung cho các điều khiển khác.

Bảng 1.2. Một số sự kiện thường dùng của Form

Sự kiện	Mô tả
Click	Xảy ra khi click vào vùng làm việc của Form
DoubleClick	Xảy ra khi click đúp vào vùng làm việc Form
FormClosed	Xảy ra sau khi hoàn thành việc đóng Form (sau khi nhấn nút X màu đỏ trên thanh tiêu đề)
FormClosing	Xảy ra khi Form đang đóng (thường dùng để ngắt kết nối CSDL)
KeyDown, KeyPress	Xảy ra khi người dùng nhấn một phím hay tổ hợp phím trên bàn phím
KeyUp	Xảy ra khi người dùng nhả một phím
MouseClick	Xảy ra khi nhấn bất kỳ phím nào của chuột
MouseDoubleClick	Xảy ra khi nhấp đôi chuột vào vùng làm việc của Form
MouseHover, MouseLeave	Xảy ra khi người dùng di chuột vào hoặc ra khỏi Form
MouseMove	Xảy ra khi người dùng di chuyển chuột trên một vùng trống trong vùng làm việc của Form
MouseUp	Xảy ra khi người dùng nhả chuột (left, right, center)
TextChanged	Xảy ra khi thay đổi thuộc tính Text của Form

Ví dụ, để thao tác khi click vào Form, màu nền sẽ chuyển từ tím sang vàng (Hình 1.7), click đôi vào sự kiện Click của frmLogin, chương trình sẽ tự sinh ra hàm xử lý sự kiện cho sự kiện này, thêm dòng code sau vào đoạn mã vừa phát sinh:

```
this.BackColor = Color.Yellow; //this là đối tượng của Form hiện hành
```



Hình 1.7. Minh họa sự kiện Click của Form

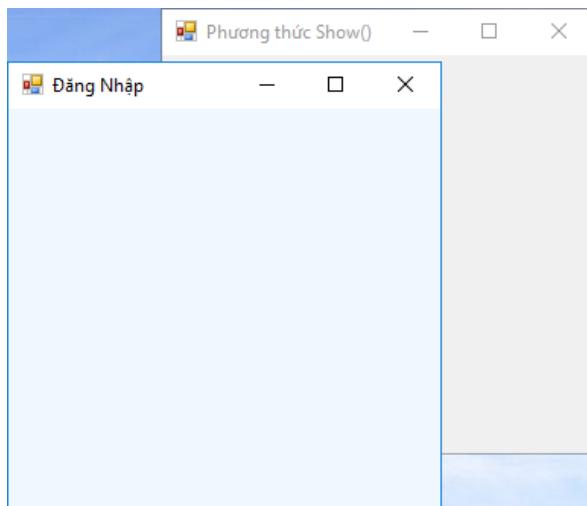
1.1.4. Các phương thức của Form

Một số phương thức thường sử dụng trong Form đó là: Show(), ShowDialog(), Hide(), Close().

- *Phương thức Show()*: Dùng để hiển thị Form mới. Khi sử dụng phương thức này để hiển thị Form mới thì người dùng vẫn có thể quay lại thao tác trên Form trước đó trong khi Form hiện hành vẫn đang hoạt động. Ví dụ để hiển thị một Form khác mới sử dụng phương thức Show(), ta thêm đoạn mã sau vào hàm sự kiện Load của Form hiện hành:

```
Form frm2 = new Form();
frm2.Text = "Phương thức Show()";
frm2.Show();
```

Kết quả chạy chương trình như Hình 1.8, lưu ý là chúng ta vẫn có thể thao tác trên Form chính ban đầu.



Hình 1.8. Minh họa phương thức Show()

- *Phương thức ShowDialog()*: Phương thức này cũng dùng để hiển thị Form. Tuy nhiên, khác biệt giữa phương thức ShowDialog() và Show() là phương thức này chỉ cho phép người dùng thao tác trên Form mới vừa hiển thị mà không thao tác được trên các Form trước đó, để thao tác được trên các Form trước thì người dùng phải đóng Form mới hiển thị. Ví dụ tạo sự kiện hiển thị một Form mới khi click vào frmLogin, sử dụng phương thức ShowDialog(), ta thêm đoạn mã sau vào sự kiện Click của frmLogin:

```
Form frm3 = new Form();
frm3.Text = "Pt ShowDialog()";
frm3.ShowDialog();
```

Kết quả chạy chương trình cũng tương tự như Hình 1.8, tuy nhiên chúng ta chỉ có thể thao tác trên Form hiện hành mà không thao tác được trên Form chính ban đầu. Nếu muốn Form hiện ra có nút OK hoặc Cancel và chúng ta làm việc với một trong hai nút này, khi đó có thể thay đổi cách gọi để kiểm tra xem người dùng có nhấn nút OK hay không, mã nguồn được thay đổi như sau:

```
Form frm3 = new Form();
frm3.Text = "Pt ShowDialog()";
if(frm3.ShowDialog() == DialogResult.OK)
{
    // Thực hiện thao tác gì đó
}
```

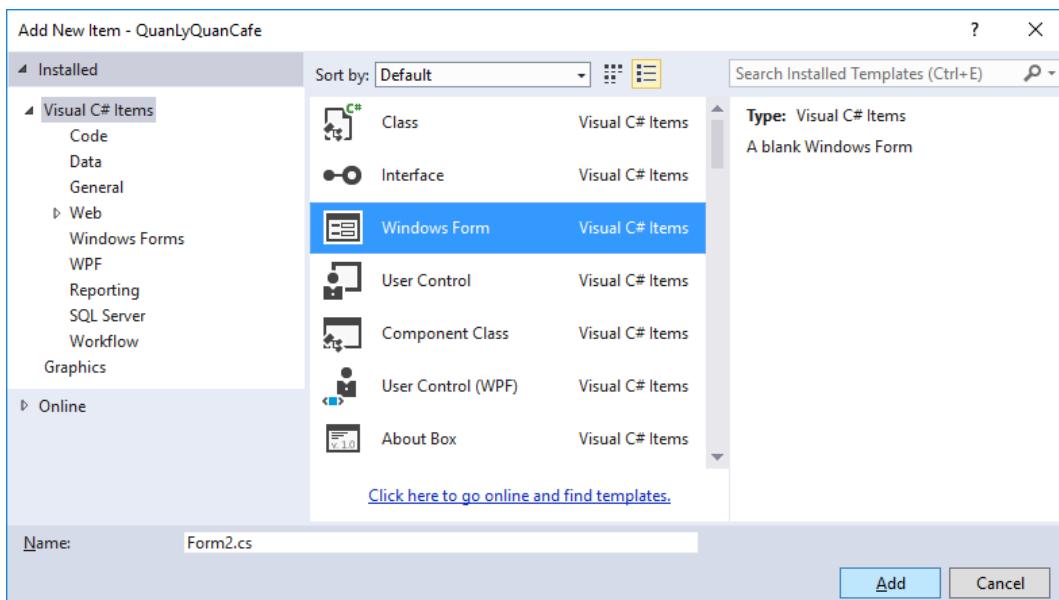
- *Phương thức Hide()*: Ẩn Form để Form không hiển thị trên màn hình. Bản chất của phương thức này là thiết lập thuộc tính *Visible = true*. Khi sử dụng

phương thức này, Form chỉ bị ẩn đi, các thông tin hoặc đối tượng xử lý trên Form vẫn có thể được truy cập.

➤ *Phương thức Close()*: Sử dụng để đóng Form sau khi thực hiện tác vụ nào đó.

1.1.5. Các loại Form

Để thêm một Form mới vào dự án: Click chuột phải vào tên dự án trong Solution Explorer chọn Add chọn New Item... hoặc bấm tổ hợp phím Ctrl+Shift+A; Tại cửa sổ Add New Item chọn Visual C# Item, chọn Windows Form sau đó đặt tên rồi bấm Add để hoàn tất việc thêm một Form mới (Hình 1.9).



Hình 1.9. Thêm một Form mới vào dự án

Visual Studio hỗ trợ ba loại Form, đó là Form thường (Normal Form), Form cha (MDI Form), và Form con (Child Form).

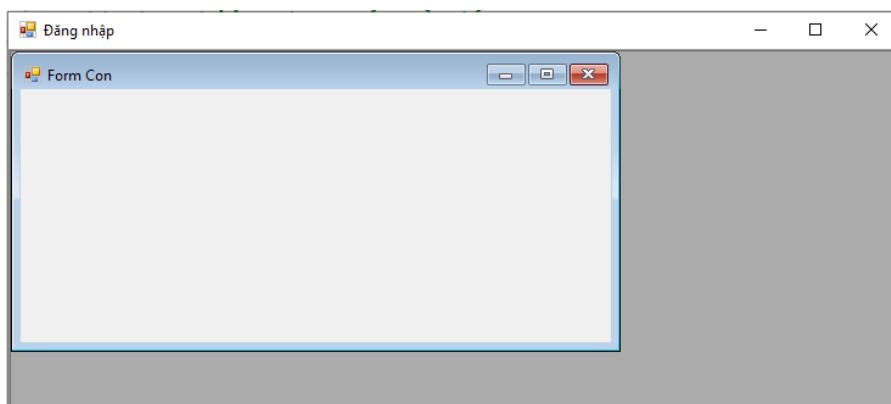
- *Form thường*: Là một Form độc lập, mặc định khi được tạo ra, nó không chứa hay được chứa bởi bất kỳ Form nào, nó không phải là Form cha cũng không phải Form con.
- *Form cha*: Là Form có thể chứa các Form khác, các Form được chứa trong nó được gọi là các Form con. Để chuyển từ một Form thường thành Form cha, ta thiết lập thuộc tính IsMdiContainer = true; Khi đó nền của Form sẽ mặc định chuyển sang màu xám đậm (Hình 1.10).
- *Form con*: Là Form nằm trong Form cha, nó chỉ hiện thị trong phạm vi của Form cha, để thay đổi một Form thường thành con của Form khác ta phải gán giá trị thuộc tính MdiParent của Form cho một đối tượng là MdiForm. Thuộc tính MdiParent không hiển thị trong Properties, ta phải thiết lập bằng mã lệnh. Ví dụ để tạo một Form con có tiêu đề “Form Con” cho frmLogin ban đầu, trước hết gán thuộc tính IsMdiContainer = true, sau đó tạo sự kiện Load cho frmLogin và thêm đoạn mã sau:

```
Form frmChild = new Form();
frmChild.Text = "Form Con";
frmChild.MdiParent = this;
frmChild.Show();
```

Kết quả chạy chương trình như trong Hình 1.11.



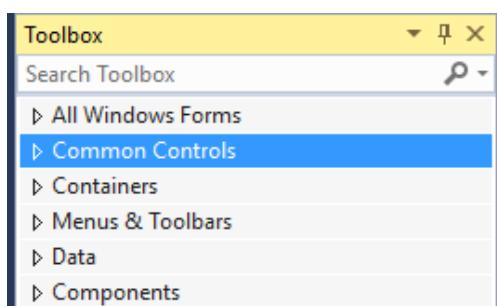
Hình 1.10. Minh họa Form cha



Hình 1.11. Minh họa mối quan hệ cha con của Form

1.2. Các điều khiển cơ bản

Danh sách các điều khiển (Controls) được cung cấp trong thanh công cụ Toolbox của giao diện làm việc Visual Studio (Hình 1.12). Các điều khiển này được liệt kê theo nhóm thể hiện như trong Bảng 1.3. Chúng ta có thể dùng nó bằng cách kéo thả vào Form (nếu thành thạo, có thể viết mã thay vì kéo thả).



Hình 1.12. Hộp thoại Toolbox trong Visual Studio

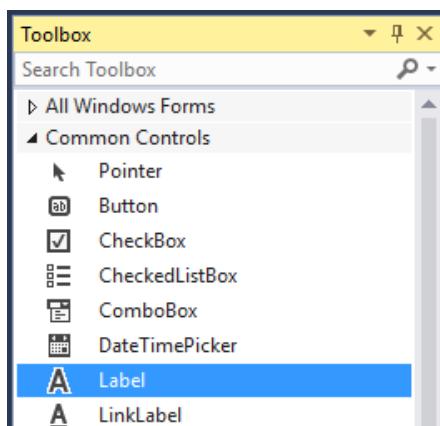
Bảng 1.3. Các nhóm điều khiển trong thanh Toolbox

Tên nhóm	Mô tả	Ví dụ
All Windows Forms	Tất cả các Controls	
Common Controls	Nhóm Control thường được sử dụng	Button, Checkbox, Label, ListView...
Containers	Nhóm Control chứa các Control khác	GroupBox, Panel...
Menus and Toolbars	Nhóm Control tạo Menu và các thanh trạng thái	MenuStrip, ToolStrip...
Data	Nhóm Control thao tác với cơ sở dữ liệu	DataGridView, DataSet...
Components	Nhóm Control dùng để kiểm tra dữ liệu nhập	EventLog, ImageList...
Printing	Nhóm Control thao tác việc xuất bản và in ấn	PrintDialog, PageSetupDialog...
Dialog	Nhóm Control xử lý hộp thoại	OpenFileDialog, ColorDialog...
Reporting	Nhóm Control xử lý báo cáo	SnapControl, DocumentViewer...
WPF Interoperability	Nhóm Control dùng trong WPF	ElementHost...
General	Nhóm Control do người dùng xây dựng	CustomControl, UserControl...

Việc bố trí các Control, Components theo nhóm giúp cho lập trình viên có thể dễ dàng tìm kiếm các điều khiển cần thiết tùy vào mục đích sử dụng. Ngoài ra, khi cài đặt thêm các công cụ khác có hỗ trợ Control, Visual Studio tự động thêm các Control vào thanh này theo nhóm.

1.2.1. Điều khiển Label

Điều khiển Label (Nhãn) (Hình 1.13) thường dùng để mô tả thông tin cho các điều khiển khác, hiển thị thông tin dưới dạng chỉ đọc (*read-only*). Label có một số thuộc tính được mô tả trong Bảng 1.4.

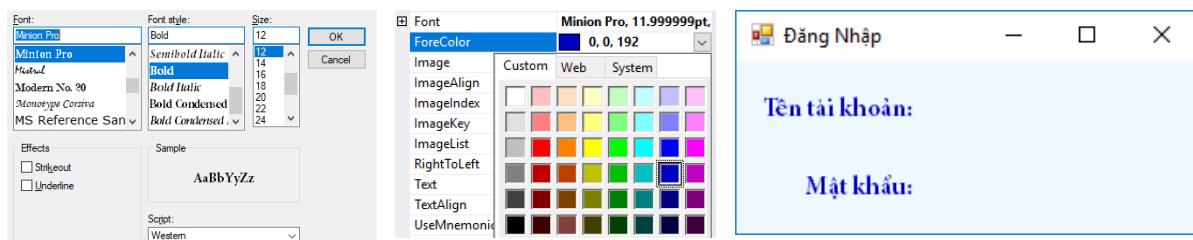


Hình 1.13. Điều khiển Label

Bảng 1.4. Một số thuộc tính thường dùng của Label

Thuộc tính	Mô tả
Name	Định danh duy nhất của Label
BorderStyle	Thiết lập đường viền
Font	Thiết lập kiểu, kích thước chữ hiển thị trên Label
Text	Nội dung hiển thị trên Label
BackColor	Màu nền của Label
ForeColor	Màu chữ hiển thị trên Label
TextAlign	Căn lề chữ hiển thị trên Label
Visible	Ẩn/Hiển thị Label

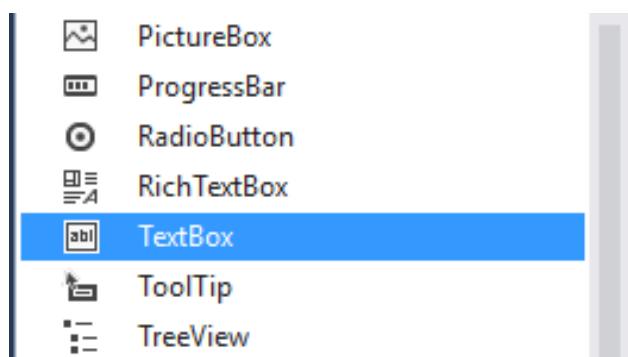
Ví dụ, để tạo hai nhãn hiển thị nội dung là “Tên tài khoản” và “Mật khẩu” cho frmLogin: Trong giao diện Form của Visual Studio, kéo hai Label từ Toolbox vào frmLogin; Bật cửa sổ Properties, thiết lập giá trị các thuộc tính cho các Label như sau: Với thuộc tính Font: Font chữ Minion Pro, kiểu chữ đậm, kích thước 12; Màu xanh (Hình 1.14).



Hình 1.14. Thiết lập các tham số cho công cụ Label

1.2.2. Điều khiển TextBox

TextBox là điều khiển cho phép người dùng nhập dữ liệu đầu vào (Hình 1.15). Người dùng có thể nhập số, chữ hoặc ký tự, Windows Forms coi dữ liệu này như là một chuỗi. Các thuộc tính và sự kiện cơ bản của TextBox được mô tả trong Bảng 1.5 và Bảng 1.6. Ngoài ra, TextBox cũng có một số phương thức xử lý cơ bản, được minh họa như trong Bảng 1.7.



Hình 1.15. Điều khiển TextBox

Bảng 1.5. Các thuộc tính thường dùng của TextBox

Thuộc tính	Mô tả
Name	Định danh duy nhất của TextBox. Để dễ thao tác, người ta thường bắt đầu bởi tiền tố txt, ví dụ: txtUser, txtPassword...
BackColor	Màu nền của TextBox
Enable	Nhận giá trị true/false, nếu là false thì không cho phép người dùng thao tác trên TextBox
Font	Thiết lập kiểu, kích thước chữ hiển thị trên TextBox
ForeColor	Màu chữ hiển thị trên TextBox
MaxLength	Quy định độ dài tối đa của chuỗi nhập vào
Multiline	Nhận giá trị true/false, nếu là true thì cho phép hiển thị nhiều dòng trên TextBox
PasswordChar	Ký tự thay thế cho dữ liệu nhập vào, dùng để che giấu dữ liệu, thường sử dụng khi người dùng nhập mật khẩu
ReadOnly	Nhận giá trị true/false, nếu là true thì không cho phép sửa đổi nội dung trên TextBox
ScrollBars	Có hiển thị thanh cuộn khi nội dung vượt quá kích thước hiển thị của TextBox hay không
Size	Kích thước của TextBox (rộng x cao)
Text	Nội dung ban đầu hiển thị trên TextBox khi khởi chạy chương trình
Visible	Ẩn/Hiển thị TextBox

Bảng 1.6. Các sự kiện thường dùng của TextBox

Sự kiện	Mô tả
Click	Xảy ra khi click chuột vào TextBox
DoubleClick	Xảy ra khi click đôi chuột vào TextBox
KeyDown	Xảy ra khi người dùng nhấn một phím bất kỳ
KeyUp	Xảy ra khi người dùng nhả một phím đã nhấn
KeyPress	Xảy ra khi người dùng nhấn và nhả một phím
MouseEnter	Xảy ra khi người dùng di chuột vào vùng của TextBox
MouseLeave	Xảy ra khi người dùng di chuột ra khỏi TextBox
MouseMove	Xảy ra khi người dùng rê chuột trên vùng của TextBox
TextChanged	Xảy ra khi người dùng gõ vào ô nhập của TextBox

Ví dụ: Để tạo giao diện cho Form frmLogin cho phép người dùng nhập Tên tài khoản và Mật khẩu, thực hiện như sau: Kéo các điều khiển TextBox vào giao diện Form và trang trí như Hình 1.16; Đặt lại tên cho các TextBox lần lượt là txtUser và txtPass (nên đặt tên TextBox gắn với chức năng cụ thể của từng TextBox để dễ dàng thao tác và xử lý); Đặt kích thước font chữ cho dữ liệu nhập vào ở các TextBox là 10; Đặt thuộc tính PasswordChar cho txtPass là dấu sao *; Chạy chương trình, nhập dữ liệu, và kiểm tra kết quả.



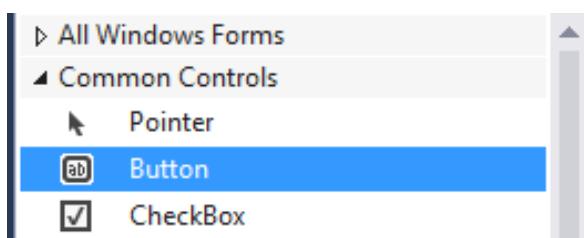
Hình 1.16. Minh họa xây dựng giao diện với TextBox

Bảng 1.7. Các phương thức cơ bản của TextBox

Phương thức	Mô tả
Clear()	Xóa toàn bộ nội dung của TextBox
Copy()	Sao chép phần nội dung được chọn trong TextBox
Cut()	Cắt đi phần nội dung được chọn trong TextBox
Paste()	Dán phần nội dung đã sao chép hoặc cắt trước đó
Undo()	Khôi phục thao tác trước
Select()	Chọn một phần nội dung của TextBox
SelectAll()	Chọn toàn bộ nội dung của TextBox
DeselectAll()	Bỏ chọn nội dung của TextBox

1.2.3. Điều khiển Button

Button cũng nằm trong nhóm Common Controls, Button là điều khiển nút bấm cho phép người dùng click vào để kích hoạt một hành động cụ thể nào đó hoặc để lựa chọn các tùy chọn của chương trình. Một chương trình có thể gồm nhiều loại Button như Button, CheckBox, hay RadioButton; Tất cả các lớp Button này đều là kế thừa từ lớp ButtonBase - nơi định nghĩa các thuộc tính chung của Button (nằm trong namespace System.Windows.Forms). Ở mục này ta đề cập đến Button (Hình 1.17), loại điều khiển thực hiện lệnh trên Form. Các thuộc tính và sự kiện của Button được mô tả trong Bảng 1.8 và Bảng 1.9.



Hình 1.17. Điều khiển Button

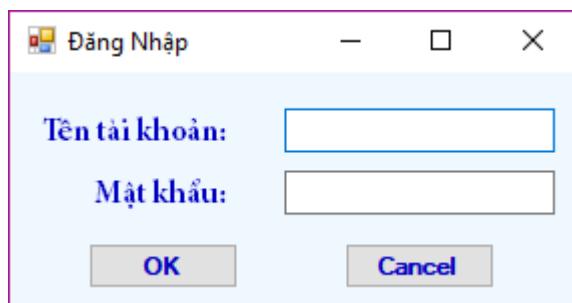
Bảng 1.8. Các thuộc tính thường dùng của Button

Thuộc tính	Mô tả
Name	Định danh duy nhất của Button. Để dễ sử dụng, thường bắt đầu bởi btn, ví dụ: btnOK, btnCancel.
BackColor	Màu nền của Button
Enable	Nhận giá trị true/false, nếu là false thì không cho phép người dùng thao tác trên Button
Font	Thiết lập kiểu, kích thước chữ hiển thị trên Button
ForeColor	Màu chữ hiển thị trên Button
Image	Hình ảnh hiển thị trên Button
Text	Nội dung ban đầu hiển thị trên Button
Visible	Ẩn/Hiển thị Button

Bảng 1.9. Các sự kiện thường dùng của Button

Sự kiện	Mô tả
Click	Xảy ra khi click chuột vào Button
MouseEnter	Xảy ra khi chuột nằm trong vùng hiển thị của Button
MouseHover	Xảy ra khi người dùng di chuột vào Button
MouseLeave	Xảy ra khi người dùng di chuyển chuột khỏi Button
MouseMove	Xảy ra khi di chuyển chuột trên Button

Ví dụ, để tạo nút OK và Cancel cho frmLogin, ta kéo các Button từ Toolbox vào giao diện frmLogin, đặt tên cho các Button lượt là btnOK và btnCancel, thiết lập thuộc tính Text tương ứng là “OK” và “Cancel”, thiết lập font chữ là: Microsoft Sans Serif, kiểu chữ đậm, kích thước 8 (Hình 1.18).



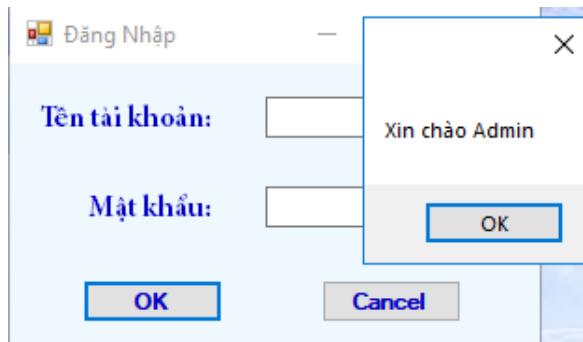
Hình 1.18. Minh họa tạo Button

Công việc tiếp theo là tạo sự kiện cho btnOK để khi bấm vào nút OK thì sẽ hiển thị ra thông báo “Xin chào Admin” (ở đây ta chỉ đơn giản là xuất ra thông báo, chưa xử lý dữ liệu mà người dùng nhập vào các TextBox). Trước tiên, click đúp chuột vào btnOK (hoặc tạo sự kiện Click của btnOK trong cửa sổ Properties) để khởi tạo code xử lý sự kiện, thêm đoạn mã sau cho sự kiện:

```
MessageBox.Show("Xin chào Admin");
```

Kết quả chạy chương trình được thể hiện như Hình 1.19, đối với btnCancel, chúng ta cũng thực hiện như thao tác trên, với lệnh:

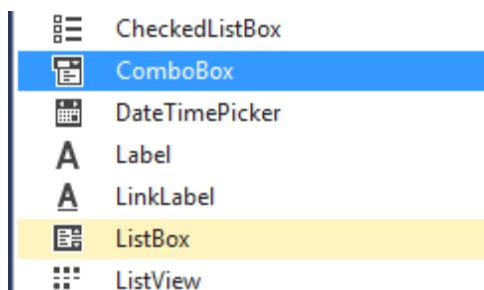
```
Application.Exit();
```



Hình 1.19. Kết quả chạy chương trình với sự kiện của btnOK

1.2.4. Điều khiển ComboBox và điều khiển ListBox

Điều khiển ComboBox và ListBox giúp cung cấp sẵn một danh sách phần tử để người dùng lựa chọn (Hình 1.20), cho phép chọn một hay nhiều phần tử. ComboBox cho phép người dùng chọn phần tử với danh sách sổ xuống, còn ListBox thì cho người dùng chọn lựa với một danh sách các phần tử được trình bày sẵn. Cả ComboBox và ListBox đều sử dụng sự kiện SelectedIndexChanged để xử lý việc lựa chọn mục chọn của người dùng. Tương tự như các điều khiển trên, hai điều khiển này cũng có những thuộc tính dùng chung (Bảng 1.10) và cũng có những thuộc tính dùng riêng cho mỗi điều khiển (Bảng 1.11 và Bảng 1.12).



Hình 1.20. Điều khiển ComboBox và ListBox trong Toolbox

Bảng 1.10. Các thuộc tính chung thường dùng của ComboBox và ListBox

Thuộc tính	Mô tả
DataSource	Tập dữ liệu nạp vào điều khiển dạng danh sách
Items	Biểu diễn tập danh sách phần tử
SelectedIndex	Trả về chỉ số của phần tử được chọn, phần tử đầu tiên có chỉ số là 0
SelectedItem	Trả về phần tử được chọn
SelectedText	Trả về chuỗi hiển thị của phần tử được chọn
SelectedValue	Trả về giá trị của phần tử được chọn

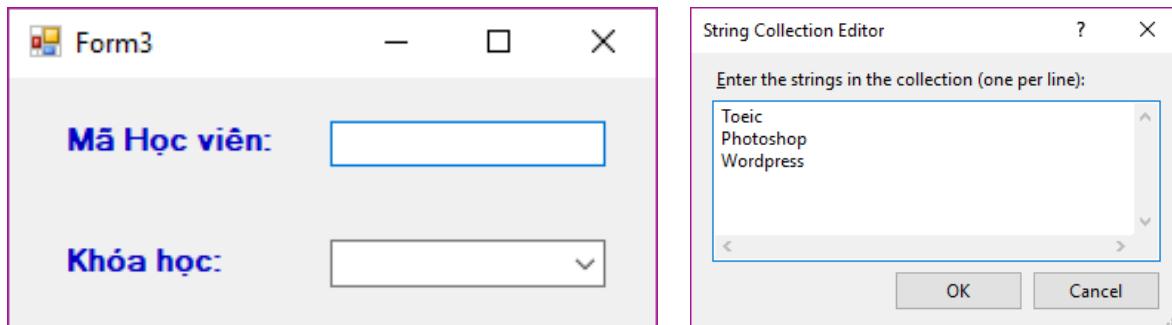
Bảng 1.11. Các thuộc tính thường dùng của ComboBox

Thuộc tính	Mô tả
Name	Tên của ComboBox. Nên đặt tên bắt đầu với tiền tố, ví dụ: cbbLop
Text	Nội dung mặc định hiển thị lên ComboBox
DropDownStyle	Quy định dạng hiển thị: DropDown: Có thể chọn hoặc nhập thêm dữ liệu mới; Simple: Hiển thị dạng liệt kê kèm ô nhập liệu để cho phép nhập; DropDownList: Chỉ cho phép chọn dữ liệu có sẵn.

Bảng 1.12. Các thuộc tính thường dùng của ListBox

Thuộc tính	Mô tả
Name	Tên của ListBox, nên bắt đầu bởi một tiền tố như lbMonHoc
MultiColumn	Cho phép hiển thị các mục thành nhiều cột hay không
SelectionMode	Quy định cách thức chọn dữ liệu: <i>One</i> : Cho phép chọn một giá trị; <i>MultiSimple</i> : Cho phép chọn nhiều giá trị bằng cách click vào mục chọn; <i>MultiExtended</i> : Chọn nhiều giá trị bằng cách click kết hợp với phím Shift hoặc Ctrl
SelectedItems	Trả về các dòng dữ liệu được chọn
SelectedIndices	Trả về các chỉ số của các dòng dữ liệu được chọn

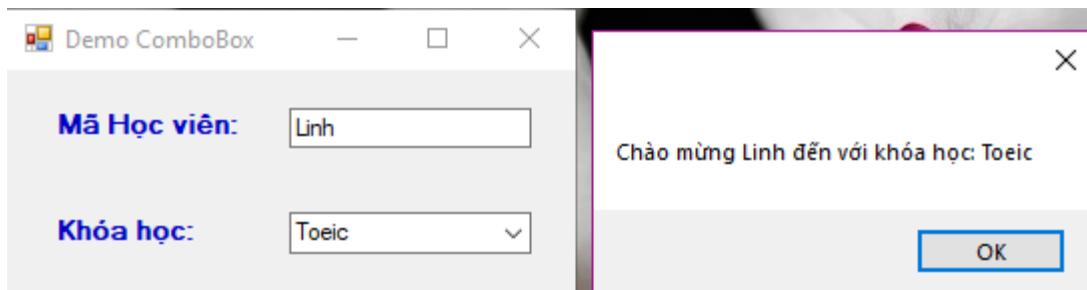
Ví dụ, để xây dựng chương trình có giao diện bao gồm một Mã học viên và một ComboBox cbbKhoaHoc chứa danh sách ba khóa học là: Toeic, Photoshop, và WordPress. Chọn ComboBox, trong cửa sổ Properties, chọn Items, và thêm dữ liệu cho ComboBox (Hình 1.21).



Hình 1.21. Minh họa giao diện với ComboBox chứa dữ liệu

Để cho phép khi người dùng chọn một khóa học thì sẽ hiện ra thông báo “Chào mừng đến với khóa học...” (Hình 1.22), click đúp chuột vào ComboBox để tạo sự kiện SelectedIndexChanged và điền đoạn mã sau cho sự kiện đó:

```
MessageBox.Show("Chào mừng " + this.textBox1.Text + " đến với khóa học: " + this.cbbKhoaHoc.SelectedItem.ToString());
```



Hình 1.22. Kết quả chạy chương trình khi người dùng chọn đối tượng trong ComboBox

Bài tập: Viết chương trình với chức năng tương tự như ví dụ trên nhưng sử dụng ListBox.

1.2.5. Điều khiển CheckBox và điều khiển RadioButton

CheckBox là điều khiển cung cấp các lựa chọn cho người dùng dưới dạng hộp kiểm, cho phép người dùng có thể chọn một hay nhiều lựa chọn cùng lúc. Trong khi đó RadioButton chỉ cho phép người dùng chọn một trong số các lựa chọn cùng nhóm, khi một lựa chọn được chọn (checked) thì các lựa chọn khác sẽ tự động bỏ chọn (unchecked). Thông thường, khi thiết kế, các RadioButton sẽ được chứa trong một điều khiển nào đó thuộc nhóm Containers (GroupBox, Panel, ...) để tạo thành một nhóm, người dùng chỉ được chọn một RadioButton trong một nhóm. Ngoài các thuộc tính chung như Name, Text như các điều khiển khác, Bảng 1.13 và Bảng 1.14 lần lượt mô tả một số thuộc tính và sự kiện thường dùng của hai điều khiển này.

Bảng 1.13. Các thuộc tính chung thường dùng của CheckBox và RadioButton

Thuộc tính	Mô tả
Checked	Nhận giá trị true hoặc false, thiết lập cho điều khiển được chọn hay không
CheckState	Chỉ có ở CheckBox, nhận một trong 3 giá trị: Checked: Điều khiển được chọn; Unchecked: Điều khiển không được chọn; và Indeterminate: Điều khiển bị vô hiệu
RightToLeft	Thiết lập hiển thị chuỗi văn bản bên trái hay bên phải của điều khiển

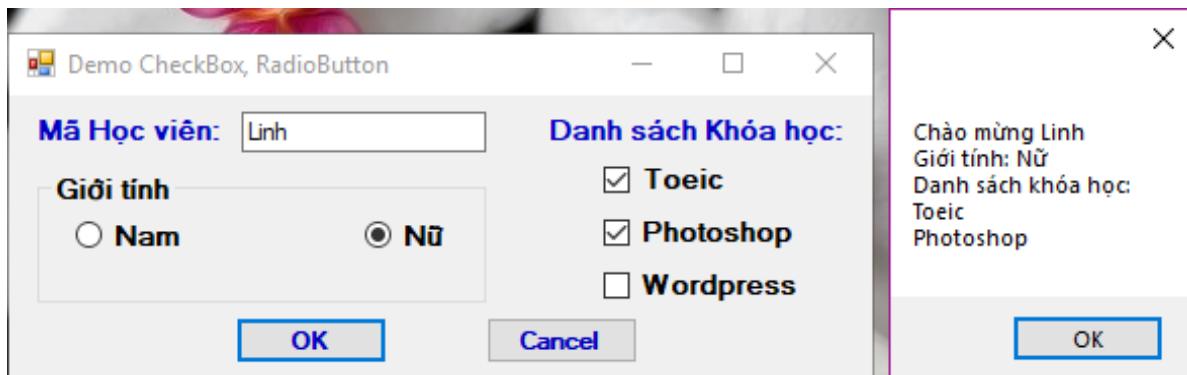
Bảng 1.14. Các sự kiện thường dùng của CheckBox và RadioButton

Sự kiện	Mô tả
Click	Xảy ra khi người dùng click chuột vào đối tượng
CheckedChange	Xảy ra khi điều khiển thay đổi từ trạng thái Checked sang Unchecked và ngược lại

Ví dụ: Xây dựng chương trình có giao diện như Hình 1.23, khi người dùng click vào btnOK thì sẽ hiện ra thông báo chào mừng. Sự kiện Click của btnOK được viết như sau:

```
private void btnOK_Click(object sender, EventArgs e)
{
    string str = "Chào mừng " + this.txtTenHV.Text + "\r\n";
    if (rdNam.Checked == true) str += "Giới tính: Nam\r\n";
}
```

```
        else
            str += "Giới tính: Nữ\r\n";
        str += "Danh sách khóa học:\r\n";
        if (ckbToeic.Checked == true) str += "Toeic\r\n";
        if (ckbPts.Checked == true) str += "Photoshop\r\n";
            if (ckbWp.Checked == true) str += "Wordpress\r\n";
        MessageBox.Show(str);
    }
```

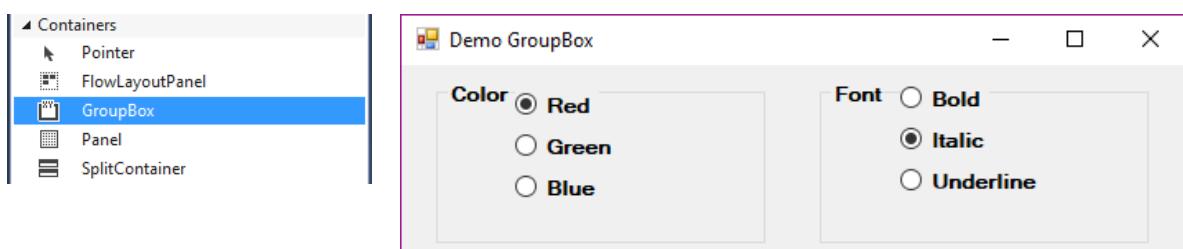


Hình 1.23. Minh họa CheckBox và RadioButton

1.3. Điều khiển chứa các điều khiển khác

1.3.1. GroupBox

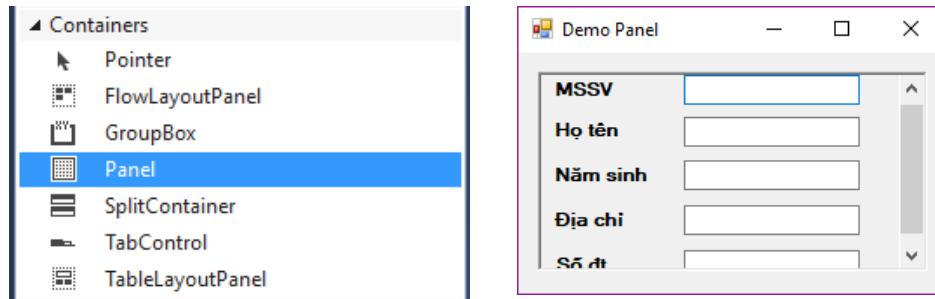
GroupBox là điều khiển thuộc nhóm Containers (Hình 1.24), là dạng điều khiển có thể chứa các điều khiển khác, giúp cho việc bố trí các điều khiển trên Form được khoa học và dễ nhìn hơn. GroupBox bản chất là một khung bao quanh các điều khiển khác, khi xóa GroupBox thì các điều khiển chứa trong nó cũng bị xóa theo. GroupBox ngoài thuộc tính thường dùng là Name và Text còn có thuộc tính Controls dùng để chứa các điều khiển khác.



Hình 1.24. Điều khiển GroupBox trong Toolbox và ví dụ minh họa

1.3.2. Panel

Tương tự như GroupBox, Panel cũng là điều khiển dùng để bố trí các điều khiển theo nhóm (Hình 1.25), điểm khác biệt giữa Panel và GroupBox là Panel không có tiêu đề, tức là không có thuộc tính Text, và Panel có chứa thanh cuộn (ScrollBar) để cho phép xem được nhiều điều khiển chứa trong nó khi kích thước của nó bị giới hạn. Bảng 1.15 mô tả một số thuộc tính thường dùng của điều khiển này.



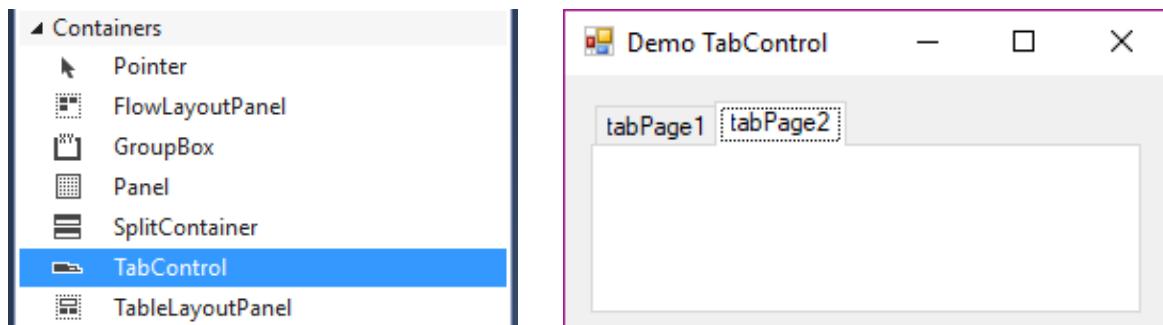
Hình 1.25. Điều khiển Panel trong Toolbox và minh họa với thuộc tính AutoScroll = true và BorderStyle = Fixed3D

Bảng 1.15. Các thuộc tính thường dùng của Panel

Thuộc tính	Mô tả
AutoScroll	Nhận giá trị true hoặc false, nếu là true thì cho phép tự động hiển thị thanh cuộn khi có nhiều điều khiển
BorderStyle	Thiết lập đường viền của Panel: None: Không có đường viền; FixedSingle: Đường viền đơn; Fixed3D: Đường viền dạng 3 chiều

1.3.3. TabControl

TabControl cũng là một điều khiển cho phép hiển thị nhiều trang trên một Form duy nhất, mỗi trang có thể chứa nhiều điều khiển. Mỗi trang chứa một Tab, khi người dùng click vào các Tab này để chuyển đổi qua lại giữa các trang. Thuộc tính quan trọng nhất của TabControl chính là TabPages, một TabControl có thể chứa nhiều TabPage, người dùng bấm vào các Tab để thao tác qua lại giữa các TabPage (Hình 1.26). Các thuộc tính chính của TabControl được mô tả như trong Bảng 1.16.



Hình 1.26. Điều khiển TabControl và minh họa với thuộc tính TabPage

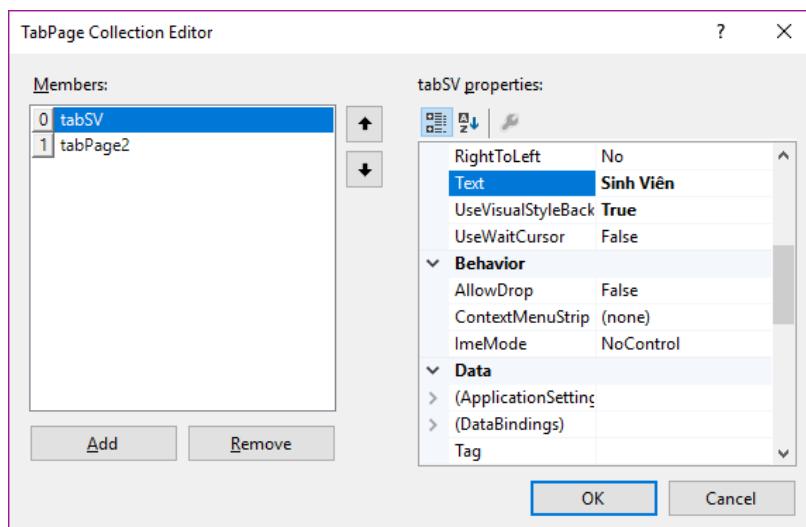
Bảng 1.16. Các thuộc tính thường dùng của TabControl

Thuộc tính	Mô tả
Alignment	Thiết lập vị trí hiển thị của các TabPage trong TabControl: Top: Hiển thị trên đầu; Bottom: Hiển thị dưới cuối; Right: Hiển thị bên phải; và Left: Hiển thị bên trái
Appearance	Thiết lập kiểu hiển thị của các TabPage, có 3 giá trị: Normal, Buttons, FlatButtons

Bảng 1.16. Các thuộc tính thường dùng của TabControl (tiếp)

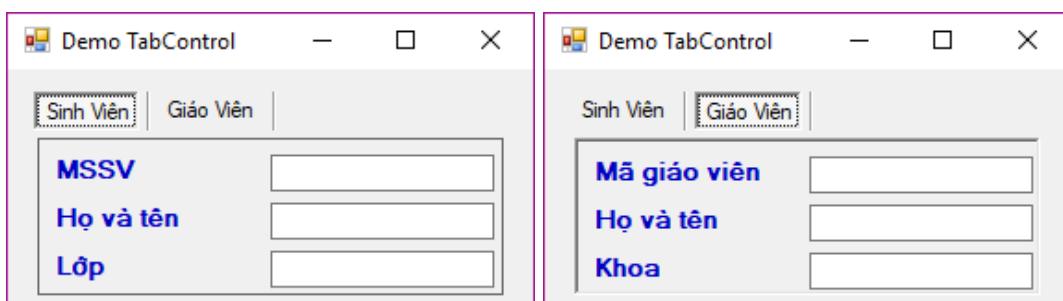
Thuộc tính	Mô tả
multiline	Nhận giá trị true hoặc false, nếu là true thì cho phép hiển thị các Tab thành nhiều dòng trong trường hợp số lượng tab vượt quá độ rộng giới hạn của TabControl
TabPage	Danh sách các TabPage chứa trong TabControl
TabCount	Trả về số lượng các TabPage chứa trong TabControl
SelectedTab	Trả về điều khiển Tab nào được chọn
SelectedIndex	Trả về vị trí của điều khiển TabPage được chọn

Mỗi TabPage trong TabControl được xem như một điều khiển Container, mỗi TabPage có các thuộc tính riêng, các thuộc tính này được thiết lập trong cửa sổ TabPage Collection Editor, thông qua thuộc tính TabPages của TabControl. Khung bên trái cửa sổ hiển thị danh sách các TabPage chứa trong TabControl, khung bên phải hiển thị danh sách các thuộc tính tương ứng của từng TabPage, lập trình viên sẽ thiết lập các thuộc tính cho từng TabPage theo yêu cầu của chương trình. Ngoài ra, có thể thêm hoặc xóa các TabPage bằng cách nhấn nút Add hoặc Remove (Hình 1.27)



Hình 1.27. Cửa sổ TabPage Collection Editor

TabPage cũng hỗ trợ thuộc tính AutoScroll và BorderStyle với chức năng tương tự như trong điều khiển Panel, ngoài ra, nó còn có thêm thuộc tính Text để hiển thị chuỗi mô tả của TabPage trên Tab (Hình 1.28).

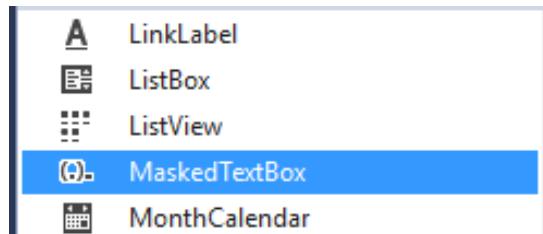


Hình 1.28. Minh họa TabControl

1.4. Một số điều khiển đặc biệt

1.4.1. MaskedTextBox

Thông thường, để người dùng nhập dữ liệu vào ta thường dùng điều khiển TextBox, tuy nhiên, khi muốn người dùng nhập dữ liệu theo một định dạng nào đó ví dụ như số điện thoại, điểm... thì không thể sử dụng TextBox vì nó không cho phép ràng buộc cách nhập liệu. Thay vào đó, chúng ta có thể sử dụng điều khiển MaskedTextBox (Hình 1.29), điều khiển này cho phép lập trình viên thiết lập quy tắc định sẵn cho dữ liệu nhập vào. Cũng như những điều khiển khác, điều khiển này cũng có những thuộc tính được nêu trong Bảng 1.17, một số dạng ký tự đặc biệt trong thuộc tính Mask được minh họa trong Bảng 1.18.



Hình 1.29. Điều khiển MaskedTextBox trong thanh Toolbox

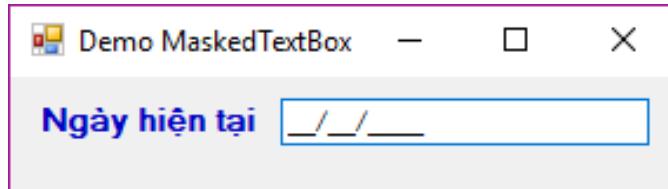
Bảng 1.17. Các thuộc tính thường dùng của MaskedTextBox

Thuộc tính	Mô tả
PromptChar	Ký tự để hiển thị khi người dùng gõ vào
TextMaskFormat	Xác định giá trị kiểu Text của điều khiển có chứa các ký tự đặc biệt và đặt chỗ hay không. Gồm có các giá trị sau: ExcludePromptAndLiterals: Không bao gồm các ký tự đặc biệt và ký tự đặt chỗ; IncludePrompt: Có chứa ký tự đặt chỗ; IncludeLiterals: Có chứa ký tự đặc biệt; và IncludePromptAndLiterals: Chứa cả ký tự đặc biệt và ký tự đặt chỗ
Mask	Thuộc tính kiểu String, xác định nguyên tắc nhập liệu cho điều khiển.

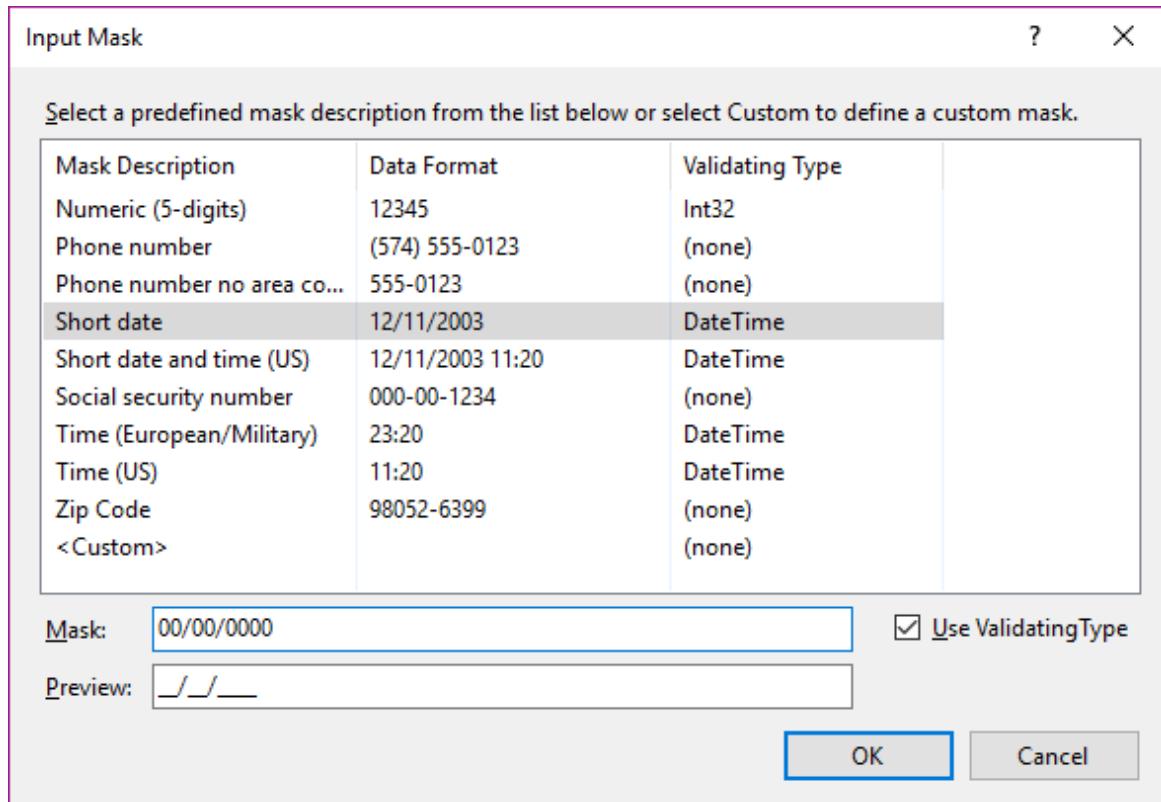
Bảng 1.18. Một số thành phần trong chuỗi thuộc tính Mask của MaskedTextBox

Thành phần	Mô tả
0	Một ký số, bắt buộc (từ 0→9)
9	Một ký số hoặc bỏ trống
#	Một ký số (hoặc dấu +, -) hoặc bỏ trống
L	Một chữ, bắt buộc (a->z, A->Z)
?	Một chữ hoặc bỏ trống
&	Một ký tự, bắt buộc
C	Một ký tự không bắt buộc
A hoặc a	Một ký tự hay ký số, không bắt buộc

Ví dụ, để thiết kế Form cho phép người dùng nhập ngày tháng theo định dạng mm/dd/yyyy (Hình 1.30), ta chọn thuộc tính Mask của MaskedTextBox và thiết lập Mask trong cửa sổ Input Mask (Hình 1.31).



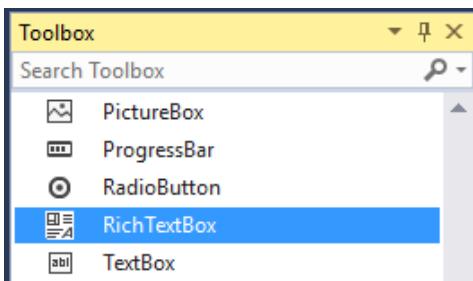
Hình 1.30. Minh họa MaskedTextBox trên Form



Hình 1.31. Thiết lập thuộc tính Mask cho MaskedTextBox

1.4.2. RichTextBox

RichTextBox là một điều khiển mở rộng từ TextBox, cho phép hiển thị dữ liệu dạng Rich Text Format (RTF), nghĩa là các đoạn text có thể được canh lề, có font chữ và màu sắc khác nhau, và có thể chứa các hình ảnh (Hình 1.32). Tương tự như những điều khiển khác, RichTextBox ngoài những thuộc tính, sự kiện cơ bản, còn có một số thuộc tính và sự kiện đặc trưng được mô tả như trong Bảng 1.19 và Bảng 1.20. Tương tự như TextBox, RichTextBox cũng có một số phương thức giúp xử lý văn bản một cách hiệu quả (Bảng 1.21).



Hình 1.32. Điều khiển RichTextBox

Bảng 1.19. Các thuộc tính thường dùng của RichTextBox

Thuộc tính	Mô tả
Font	Thiết lập Font chữ của RichTextBox
ScrollBars	Thiết lập các thanh cuộn ngang, dọc
SelectedText	Đoạn text được chọn
SelectionFont	Font chữ của đoạn text được chọn
SelectionLength	Số ký tự của đoạn text được chọn
WordWrap	Thiết lập tự động xuống dòng khi số ký tự vượt quá chiều dài của RichTextBox

Bảng 1.20. Các sự kiện thường dùng của RichTextBox

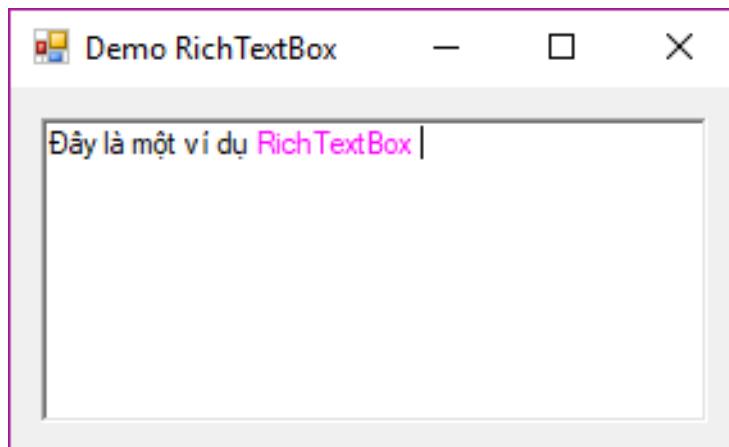
Sự kiện	Mô tả
Click	Xảy ra khi click vào RichTextBox
HScroll	Xảy ra khi người dùng click vào thanh cuộn ngang
VScroll	Xảy ra khi người dùng click vào thanh cuộn dọc

Bảng 1.21. Các phương thức thường dùng của RichTextBox

Phương thức	Mô tả
AppendText	Thêm văn bản vào RichTextBox tại vị trí con trỏ
Copy	Copy đoạn văn bản đã chọn
Paste	Dán đoạn văn bản đã copy
Redo	Trở lại thao tác đã Undo
Undo	Trở lại thao tác edit trước

Ví dụ, để thiết kế Form chứa RichTextBox (có tên là rtbThongTin), để hiển thị giao diện như Hình 1.33 ta thiết lập sự kiện Load cho của Form như sau:

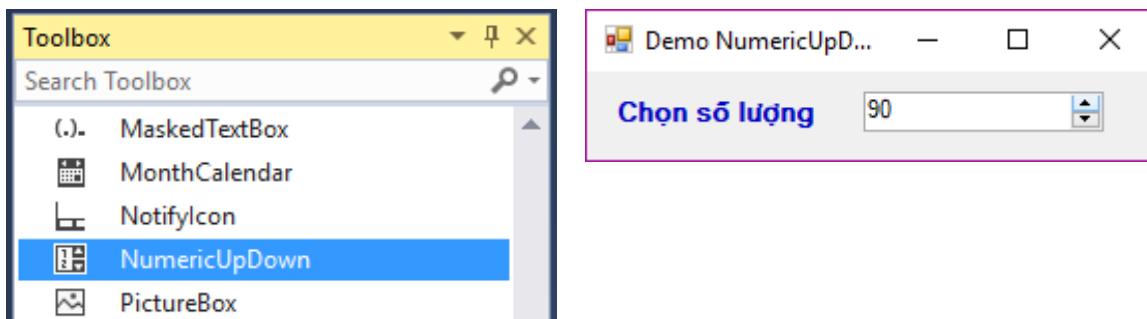
```
private void frmrtb_Load(object sender, EventArgs e)
{
    this.rtbThongTin.SelectedText = "Đây là một ví dụ ";
    this.rtbThongTin.SelectionColor = Color.Fuchsia;
    this.rtbThongTin.SelectedText = "RichTextBox";
}
```



Hình 1.33. Minh họa RichTextBox

1.4.3 NumericUpDown và DomainUpDown

NumericUpDown là điều khiển cho phép người dùng nhập hay chọn các giá trị số trong một khoảng xác định thông qua hai nút Up và Down (Hình 1.34). Một số thuộc tính cơ bản của NumericUpDown được mô tả trong Bảng 1.22. NumericUpDown sử dụng sự kiện ValueChanged để xử lý sự kiện xảy ra khi người dùng thay đổi giá trị.

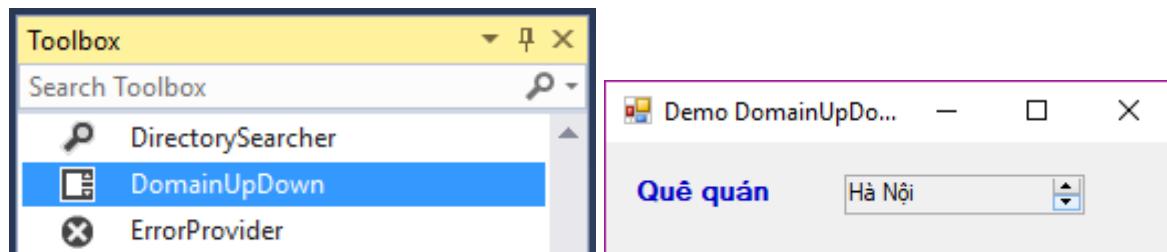


Hình 1.34. Điều khiển NumericUpDown và minh họa

Bảng 1.22. Các thuộc tính thường dùng của NumericUpDown

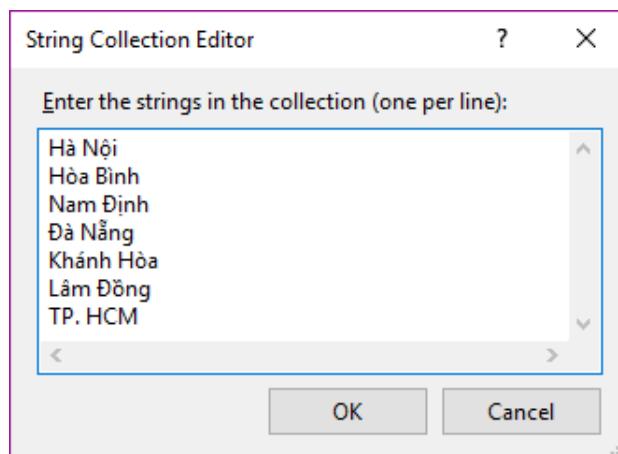
Thuộc tính	Mô tả
Value	Giá trị số hiển thị trên NumericUpDown, mặc định là 0
Maximum	Giá trị lớn nhất (mặc định là 100)
Minimum	Giá trị nhỏ nhất (mặc định là 0)
Increment	Đơn vị tăng hoặc giảm khi nhấn nút (mặc định là 1)
Maximum	Giá trị số hiển thị trên NumericUpDown, mặc định là 0

Khác với NumericUpDown, DomainUpDown là điều khiển cho phép người dùng nhập hay chọn các giá trị là các mục trong một danh sách xác định thông qua nút Up và nút Down (Hình 1.35). DomainUpDown sử dụng sự kiện SelectedIndexChanged để xử lý sự kiện xảy ra khi người dùng thay đổi giá trị trên DomainUpDown. DomainUpDown cũng có một số thuộc tính chính được mô tả tại Bảng 1.23.



Hình 1.35. Điều khiển DomainUpDown và minh họa

Ví dụ, để thiết kế Form như Hình 1.35, đặt thuộc tính `ReadOnly` của `DomainUpDown` bằng `true`, thiết lập danh sách các mục là tên các tỉnh thành trong cửa sổ `String Collection Editor` thông qua thuộc tính `Items` (Hình 1.36)



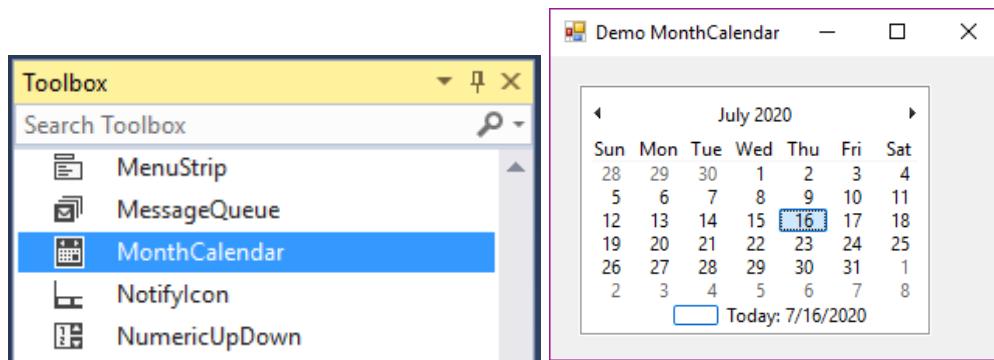
Hình 1.36. Thiết lập danh sách mục cho DomainUpDown

Bảng 1.23. Các thuộc tính thường dùng của DomainUpDown

Thuộc tính	Mô tả
Items	Danh sách các mục chọn
ReadOnly	Nhận giá trị <code>true</code> hoặc <code>false</code> , nếu là <code>true</code> thì chỉ cho phép thay đổi giá trị qua nút Up và Down mà không được phép nhập giá trị khác.
SelectedIndex	Chỉ số của mục được chọn
SelectedItem	Mục được chọn
Sorted	Sắp xếp danh sách mục

1.4.4. MonthCalendar và DateTimePicker

Điều khiển `MonthCalendar` biểu diễn một giao diện trực quan dạng lưới chứa các ngày trong tháng được biểu diễn theo tuần theo thời gian thực, cho phép người dùng chọn một ngày hay một vài ngày bất kỳ trong giới hạn thời gian thiết lập sẵn (Hình 1.37). Một số thuộc tính của `MonthCalendar` được mô tả trong Bảng 1.24. Điều khiển `MonthCalendar` thường sử dụng sự kiện `DateSelected` và `DateChanged` để xử lý hành động khi người dùng click chọn ngày.

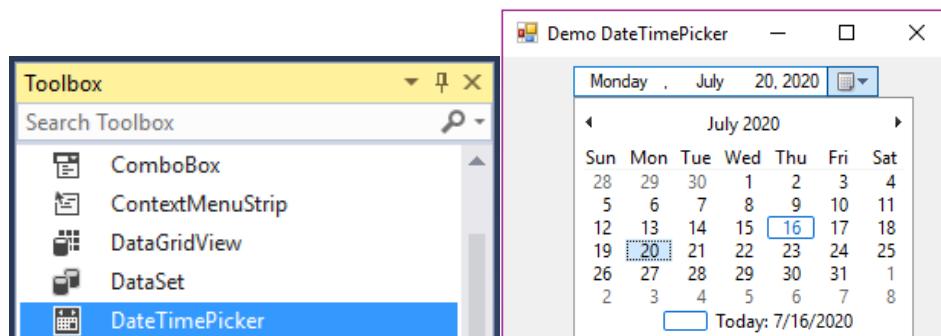


Hình 1.37. Điều khiển MonthCalendar trong thanh ToolBox và ví dụ minh họa

Bảng 1.24. Các thuộc tính thường dùng của MonthCalendar

Thuộc tính	Mô tả
SelectionStart	Ngày bắt đầu chọn
SelectionEnd	Ngày kết thúc
MaxDate	Thiết lập giá trị lớn nhất được phép nhập, chọn trên điều khiển
MinDate	Thiết lập giá trị nhỏ nhất được phép nhập, chọn trên điều khiển
MaxSelectionCount	Thiết lập số ngày tối đa người dùng được phép chọn

Tương tự, điều khiển DateTimePicker cho phép người dùng chọn ngày trong khoảng xác định thông qua giao diện đồ họa dạng lịch. Giao diện của DateTimePicker là sự kết hợp của hai điều khiển là ComboBox và MonthCalendar. DateTimePicker biểu diễn ngày tháng bởi các đối tượng thuộc lớp DateTime (Hình 1.38). Một số thuộc tính của DateTimePicker được mô tả trong Bảng 1.25. Điều khiển DateTimePicker thường sử dụng sự kiện ValueChanged để xử lý hành động khi người dùng click chọn.



Hình 1.38. Điều khiển DateTimePicker

Bảng 1.25. Các thuộc tính thường dùng của DateTimePicker

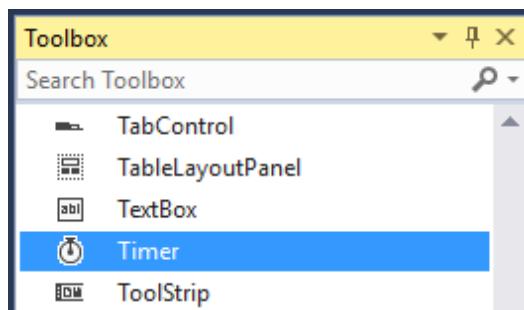
Thuộc tính	Mô tả
Format	Thiết lập kiểu hiển thị ngày tháng trong ComboBox, gồm các kiểu như Long, Short, Time, Custom
CustomFormat	Thiết lập định dạng hiển thị riêng theo ý muốn của lập trình viên. Khi sử dụng định dạng này thì thuộc tính Format phải thiết lập là Custom (Bảng 1.26)
Value	Trả về giá trị ngày hiện tại đang chọn

Bảng 1.26. Một số thành phần trong chuỗi CustomFormat của DateTimePicker

Thành phần	Mô tả
ddd	Biểu diễn 3 ký tự đầu của ngày trong tuần, ví dụ: Sun, Mon
dddd	Biểu diễn tên ngày trong tuần, ví dụ: Sunday
hh	Biểu diễn 2 ký tự giờ theo định dạng 12h
HH	Biểu diễn 2 ký tự giờ theo định dạng 24h
mm	Biểu diễn 2 ký tự phút
MM	Biểu diễn 3 ký tự đầu của tháng, ví dụ: Jan, Feb
MMM	Biểu diễn tên của tháng, ví dụ: January
ss	Biểu diễn 2 ký tự giây
t	Biểu diễn ký tự A hoặc P (Trong AM, PM)
tt	Biểu diễn AM hoặc PM
yy	Biểu diễn 2 ký tự năm, ví dụ: 19
yyyy	Biểu diễn 4 ký tự năm, ví dụ: 2019

1.4.5. Timer và ProgressBar

Điều khiển Timer cho phép thực thi hành động nào đó sau một khoảng thời gian xác định thông qua sự kiện Tick (Hình 1.39). Nghĩa là Timer sẽ tick một cách tự động (theo số miligiây của thuộc tính Interval) khi đã gọi phương thức Start(), khi Timer tick, người dùng có thể cho chương trình thực hiện một công việc nào đó. Một số thuộc tính và phương thức của Timer được mô tả trong bảng 1.27.

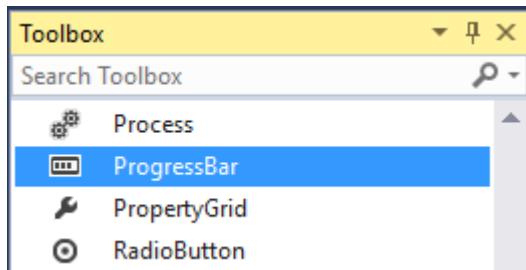


Hình 1.39. Điều khiển Timer

Bảng 1.27. Các thuộc tính và phương thức của Timer

Thuộc tính/ Phương thức	Mô tả
Interval	Thời gian kích hoạt sự kiện Tick, là số nguyên, tính bằng đơn vị mili giây (1000 interval = 1 giây)
Start()	Kích hoạt điều khiển Timer, tương ứng với việc thiết lập Enable = true
Stop()	Dừng hoạt động của Timer, tương ứng với việc thiết lập Enable = false

Điều khiển ProgressBar dùng để hiển thị tiến độ thực hiện của một công việc nào đó (Hình 1.40), điều khiển này thường được kết hợp với Timer. Một số phương thức và thuộc tính được mô tả như trong Bảng 1.28.

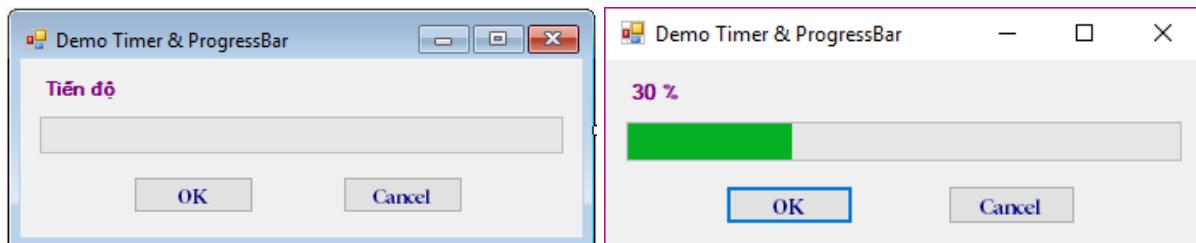


Hình 1.40. Điều khiển ProgressBar trên thanh Toolbox

Bảng 1.28. Các thuộc tính và phương thức thường dùng của ProgressBar

Thuộc tính/ Phương thức	Mô tả
Maximum	Thiết lập giá trị lớn nhất cho ProgressBar
Minimum	Thiết lập giá trị nhỏ nhất cho ProgressBar
Value	Giá trị hiện tại của thanh ProgressBar
Step	Thiết lập giá trị thêm vào Value khi gọi phương thức PerformStep()
PerformStep()	Thực hiện tăng tiến trình của ProgressBar theo giá trị thiết lập trong thuộc tính Step
Increment(int value)	Tăng vị trí hiện tại của tiến trình ProgressBar với giá trị xác định (tham số value)

Ví dụ, để thiết kế Form như Hình 1.41, thiết lập để khi bấm vào nút OK thì ProgressBar sẽ chạy theo Step được thiết lập là 10, Minimum = 0, Maximum = 20. Tiến độ chạy của ProgressBar sẽ được thể hiện trên Label thông qua sự kiện Tick của Timer, với Interval của Timer là 1000.



Hình 1.41. Minh họa ProgressBar

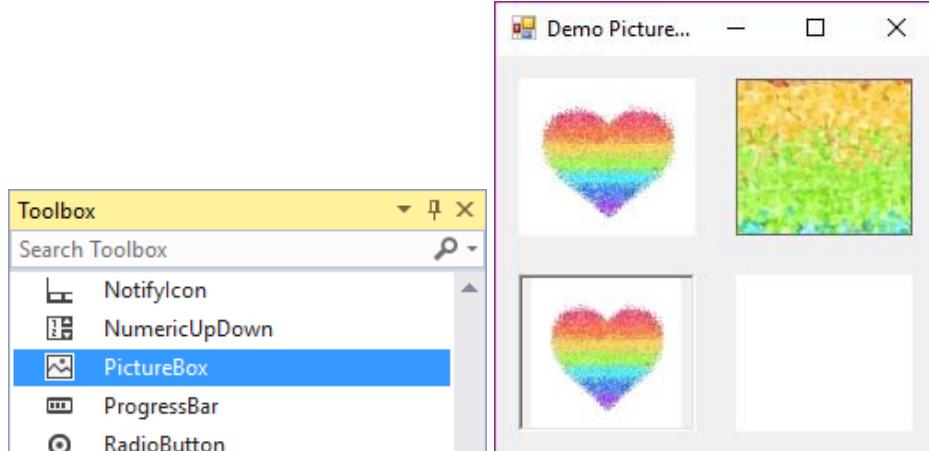
Đoạn code thực thi như sau:

```
private void btnCancel_Click(object sender, EventArgs e)
{
    this.Close();
}
```

```
private void btnOK_Click(object sender, EventArgs e)
{
    timer1.Start();
}
private void timer1_Tick(object sender, EventArgs e)
{
    if (progressBar1.Value == progressBar1.Maximum)
    {
        timer1.Stop();
        MessageBox.Show("Đã chạy xong!!!");
        progressBar1.Value = progressBar1.Minimum;
        this.lblValue.Text = "0 %";
    }
    else
    {
        progressBar1.PerformStep();
        this.lblValue.Text = progressBar1.Value.ToString() + " %";
    }
}
```

1.4.6. PictureBox và ImageList

PictureBox là điều khiển dùng để hiển thị dữ liệu dạng hình ảnh như: Bitmap, icon, jpeg, gif... . Điều khiển này chủ yếu sử dụng thuộc tính Image để thiết lập hình ảnh khi thiết kế và chạy chương trình (Hình 1.42). Bảng 1.29 giới thiệu một số thuộc tính cơ bản của điều khiển này.

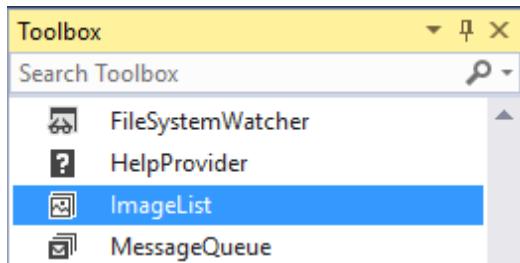


Hình 1.42. Điều khiển PictureBox trên thanh Toolbox và minh họa

Bảng 1.29. Các thuộc tính thường dùng của PictureBox

Thuộc tính	Mô tả
Image	Thiết lập hình hiển thị lên PictureBox
BorderStyle	Thiết lập kiểu viền hình ảnh
SizeMode	Thiết lập các dạng hiển thị hình ảnh

ImageList là một điều khiển đặc biệt, là một tập hợp hình ảnh có kích thước xác định để cung cấp cho các điều khiển khác sử dụng, ví dụ như: Button, ComboBox, ListView, TreeView... (Hình 1.43).

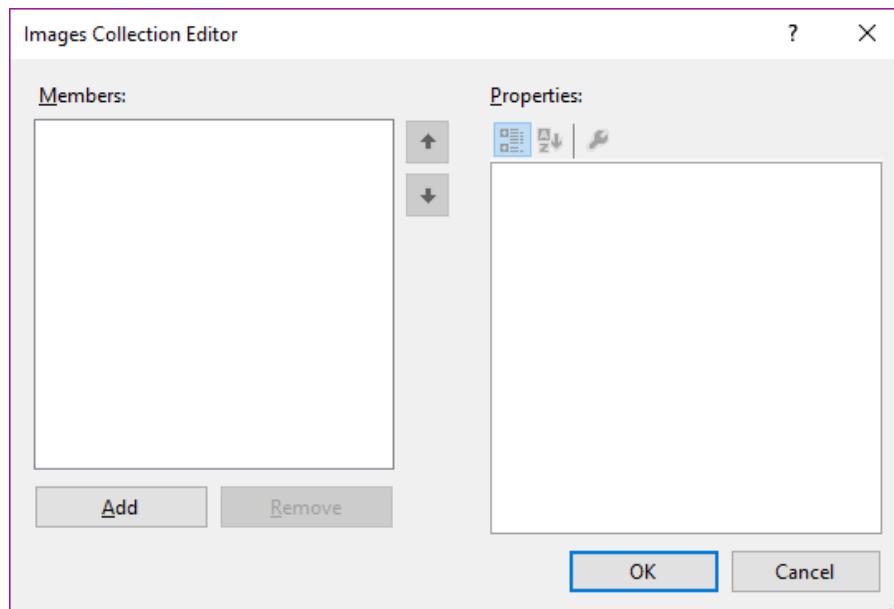


Hình 1.43. Điều khiển ImageList

Lập trình viên tiến hành thêm hoặc xóa hình ảnh trong ImageList thông qua cửa sổ Images Collection Editor, bằng cách click chọn Add hoặc Remove. Chọn thuộc tính Images của điều khiển ImageList để hiển thị cửa sổ Images Collection Editor (Hình 1.44). Một số thuộc tính thường dùng trong ImageList được mô tả trong Bảng 1.30.

Bảng 1.30. Các thuộc tính thường dùng của ImageList

Thuộc tính	Mô tả
Images	Tập các hình chứa trong ImageList (ImageList.ImageCollection)
ImageSize	Thiết lập kích thước của hình ảnh
TransparentColor	Thiết lập màu trong suốt của điều khiển



Hình 1.44. Cửa sổ Images Collection Editor của ImageList

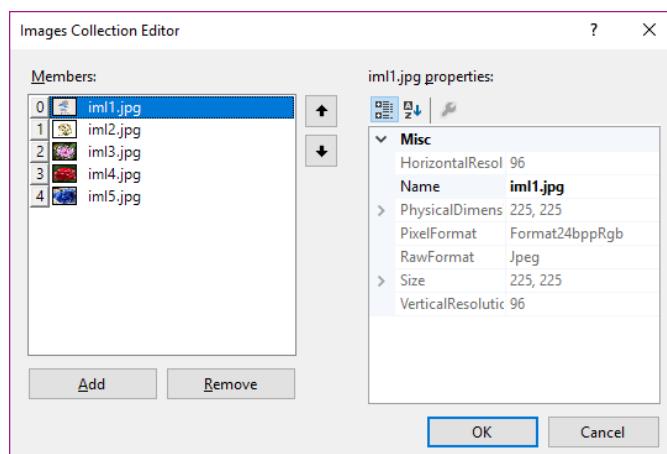
Để sử dụng ImageList cho một điều khiển nào đó, ta khai báo thuộc tính Image của điều khiển này là ImageList đã tạo, sau đó thiết lập các item/node của điều khiển này với các ImageIndex tương ứng trong ImageList. Ví dụ, xây dựng Form như Hình 1.45: Khi người dùng chọn hình muốn hiển thị chèn trong ComboBox thì hình tương

ứng sẽ xuất hiện trong PictureBox. Kéo các điều khiển ComboBox, PictureBox, và ImageList vào giao diện, thiết lập các thuộc tính cho các điều khiển như sau:

- *ComboBox*: Name = cbbChonHinh, DropDownStyle = DropDownList;
- *PictureBox*: Name = pbHinhAnh, BorderStyle = Fixed3D, Size = 256,256,SizeMode = StretchImage;
- *ImageList*: Name = imlDemo, ImageSize = 256,256, thuộc tính Images chứa một số hình ảnh như trong Hình 1.46.



Hình 1.45. Minh họa ImageList với Picture và Combobox



Hình 1.46. Thiết lập danh sách hình ảnh cho điều khiển imlDemo

Tiếp theo, xây dựng sự kiện Load của Form như sau:

```
private void frmiml_Load(object sender, EventArgs e)
{
    for(int i = 1; i <= imlDemo.Images.Count; i++)
    {
        cbbChonHinh.Items.Add("Hình " + i);
    }
}
```

Tạo sự kiện SelectedIndexChanged của điều khiển cbbChonHinh để tải hình lên pbHinhAnh, mã nguồn như sau:

```
private void cbbChonHinh_SelectedIndexChanged(object sender,
EventArgs e)
{
    pbHinhAnh.Image = imlDemo.Images[cbbChonHinh.SelectedIndex];
}
```

Kết quả thể hiện như trong Hình 1.47, khi người dùng chọn hình nào, hình đó sẽ xuất hiện trong PictureBox.



Hình 1.47. Kết quả chạy chương trình minh họa với ImageList

1.4.7. Lớp MessageBox

Lớp MessageBox là lớp dùng để hiển thị hộp thoại chứa thông báo hoặc hướng dẫn cho người dùng khi sử dụng chương trình. Người dùng bắt buộc phải thao tác trên hộp thoại trước thì mới có thể tiếp tục sử dụng chương trình. Lớp MessageBox bao gồm bốn thành phần:

- *Caption*: Chuỗi hiển thị trên thanh tiêu đề của hộp thoại;
- *Text*: Nội dung của thông báo hoặc hướng dẫn hiển thị lên cho người dùng;
- *Button*: Các nút bấm hiển thị trên hộp thoại;
- *Icon*: Biểu tượng hiển thị kèm theo thông báo/hướng dẫn.

MessageBox sử dụng phương thức Show() để hiển thị hộp thoại thông báo, kết quả trả về là DialogResult. Một số tham số của phương thức Show() được mô tả như trong Bảng 1.31.

Bảng 1.31. Mô tả các tham số của phương thức Show()

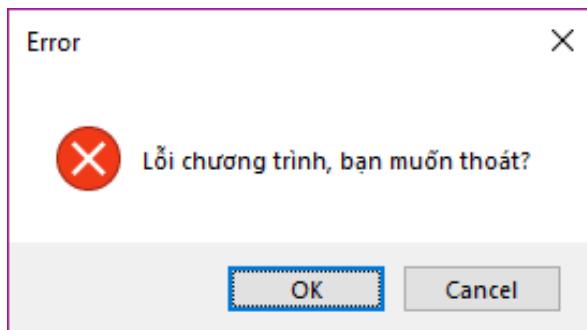
Tham số	Mô tả
MessageBoxButtons	Kiểu hiển thị nút bấm tương ứng với từng lựa chọn: OK; OKCancel; AbortRetryIgnore; YesNoCancel YesNo; RetryCancel.
MessageBoxIcon	Kiểu hiển thị các kiểu biểu tượng trên hộp thoại: Error: Hiển thị Icon dạng lỗi; Information: Hiển thị Icon thông tin; Question: hiển thị Icon câu hỏi; và Warning: Hiển thị Icon dạng cảnh báo.
MessageBoxDefaultButton	Thiết lập nút chọn mặc định trên hộp thoại: Button1: Nút chọn mặc định là nút 1; Button2: Nút chọn mặc định là nút 2; Button3: Nút chọn mặc định là nút 3

Các dạng truyền tham số cho phương thức Show thường dùng được minh họa như sau:

```
public static DialogResult Show(string text);
public static DialogResult Show(string text, string caption);
public static DialogResult Show(string text, string caption,
                               MessageBoxButtons buttons);
public static DialogResult Show(string text, string caption,
                               MessageBoxButtons buttons, MessageBoxIcon icon);
public static DialogResult Show(string text, string caption,
                               MessageBoxButtons buttons, MessageBoxIcon icon,
                               MessageBoxDefaultButton defaultButton,
                               MessageBoxOptions option);
```

Ví dụ: Để hỏi người dùng có muốn thoát chương trình hay không, nếu người dùng nhấn OK thì thoát (Hình 1.48), đoạn mã được viết như sau:

```
if(MessageBox.Show("Lỗi chương trình, bạn muốn thoát?", "Error",
MessageBoxButtons.OKCancel, MessageBoxIcon.Error)== DialogResult.OK)
{
    Application.Exit();
}
```

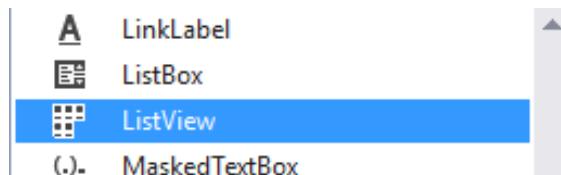


Hình 1.48. Ví dụ MessageBox với nút OK và Cancel

1.5. Một số điều khiển nâng cao

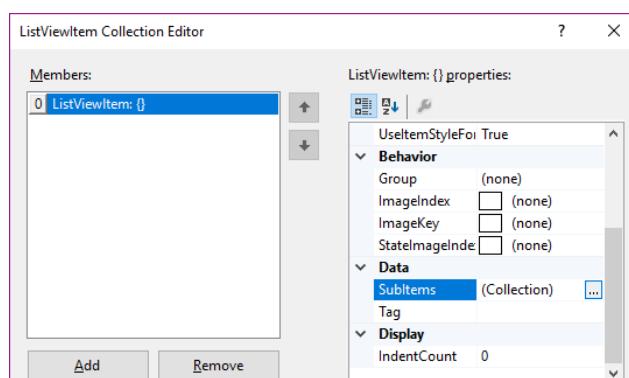
1.5.1. Điều khiển ListView

ListView là điều khiển cho phép hiển thị các đối tượng theo một danh sách, mỗi đối tượng này là một Item thuộc lớp ListViewItem, mỗi Item có thể có các Item con gọi là SubItem (Hình 1.49). ListView có thể được sử dụng để hiển thị các dữ liệu trong một bảng của cơ sở dữ liệu (Xem các chương sau). ListView có một số thuộc tính, phương thức và sự kiện được mô tả trong các Bảng 1.32, Bảng 1.33, và Bảng 1.34.

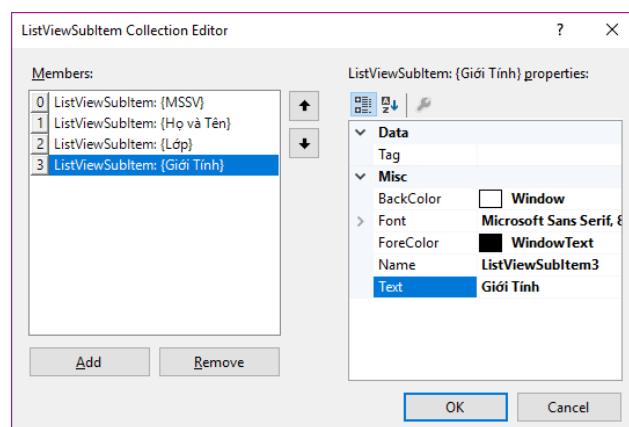


Hình 1.49. Điều khiển ListView trên thanh Toolbox

Để thêm các Item vào ListView, click chọn thuộc tính Items, cửa sổ ListViewItem Collection Editor sẽ hiện ra như Hình 1.48, chúng ta thêm hoặc xóa các Item bằng cách click Add hoặc Remove ở bên trái cửa sổ. Bên phải cửa sổ hiển thị danh sách các thuộc tính của ListViewItem. Để thêm SubItem cho một ListViewItem nào đó, click chọn thuộc tính SubItems của ListViewItem tương ứng, sau đó tiến hành thêm/xóa các SubItem trong cửa sổ ListViewSubItem Collection Editor (Hình 1.50).



Hình 1.50. Cửa sổ ListViewItem Collection Editor



Hình 1.51. Cửa sổ ListViewSubItem Collection Editor

Bảng 1.32. Các thuộc tính thường dùng của ListView

Thuộc tính	Mô tả
View	Thiết lập kiểu hiển thị của các Item trong ListView bao gồm: Details, LargeIcon, SmallIcon, List, và Tile
Items	Tập các Item chứa trong ListView
CheckBoxes	Nhận giá trị true hoặc false để thiết lập có hiển thị CheckBox hay không
GridLines	Nhận giá trị true hoặc false để thiết lập có hiển thị lưới khi thuộc tính View = Details hay không
SelectedItems	Tập hợp các mục mà người dùng đã chọn
LargeImageList/ SmallImageList	Thiết lập nguồn hình ảnh (ImageList) sử dụng cho các Item trên ListView
MultiSelect	Nhận giá trị true hoặc false để thiết lập có cho phép chọn nhiều Item một lúc hay không
FullRowSelect	Nhận giá trị true hoặc false để cho phép chọn toàn dòng dữ liệu khi click vào một ô hay không

Bảng 1.33. Các phương thức thường dùng của ListView

Phương thức	Mô tả
Clear()	Xóa toàn bộ các Items trong ListView
Sort()	Sắp xếp danh sách Item trong ListView
GetItemAt()	Lấy một Item trong ListView theo tọa độ, thông qua sự kiện Click chuột

Bảng 1.34. Các sự kiện thường dùng của ListView

Sự kiện	Mô tả
SelectedIndexChanged	Sự kiện phát sinh khi người dùng chọn chỉ mục của Item trên ListView
ItemSelectionChanged	Sự kiện phát sinh khi người dùng chọn Item trên ListView
ItemCheck/ItemChecked	Xảy ra khi trạng thái chọn của Item thay đổi
ColumnClick	Sự kiện phát sinh khi người dùng click vào một Column trên ListView
MouseClick	Sự kiện phát sinh khi người dùng click vào một Item trên ListView

Tương tự như ListView, Lớp ListViewItem cũng cung cấp một số thuộc tính và phương thức được mô tả như trong Bảng 1.35.

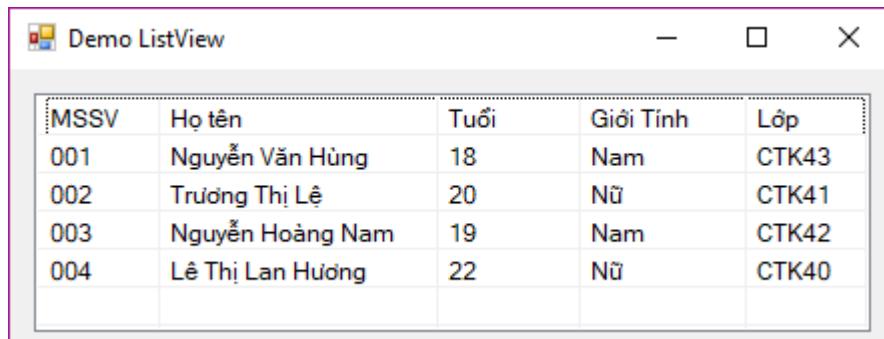
Bảng 1.35. Các thuộc tính và phương thức thường dùng của ListViewItem

Thuộc tính/ Phương thức	Mô tả
BackColor/ForeColor	Thiết lập màu nền/ màu chữ của ListViewItem
Checked	Trạng thái có được chọn hay không khi thuộc tính CheckBoxes của ListViewItem = true
Focused	Chuột có đê trên đó hay không

Bảng 1.35. Các thuộc tính và phương thức thường dùng của ListViewItem

Thuộc tính/ Phương thức	Mô tả
Selected	Trạng thái có được chọn hay không khi thuộc tính CheckBoxes của ListView = true
Text	Chuỗi hiển thị ở cột đầu tiên
SubItem	Các mục con của ListViewItem
Count()	Đếm số lượng Item trong ListView
Insert(index)	Chèn Item mới vào vị trí i trong ListView
Add()	Thêm một Item mới vào cuối ListViewItem
Remove()	Xóa Item khỏi ListView
RemoveAt(index)	Xóa Item theo chỉ số index trong ListView
IndexOf(item)	Tìm chỉ số của một Item trong ListView

Ví dụ: Thiết kế Form và chạy cho kết quả như hình 1.50, chúng ta thiết kế với một ListView: Đặt thuộc tính GridLines=true, View= Details, thêm các cột cho ListView bằng cách dùng thuộc tính Columns.



Hình 1.52. Minh họa sử dụng ListView

Tạo sự kiện FormLoad để thêm các Item vào ListView như sau, chạy chương trình để xem kết quả như Hình 1.52.

```
private void frmHocSinh_Load(object sender, EventArgs e)
{
    ListViewItem lvitem = new ListViewItem(new string[] { "001",
    "Nguyễn Văn Hùng", "18", "Nam", "CTK43" });
    this.lvDanhSach.Items.Add(lvitem);
    lvitem = new ListViewItem(new string[] { "002", "Trương Thị Lê",
    "20", "Nữ", "CTK41" });
    this.lvDanhSach.Items.Add(lvitem);
    lvitem = new ListViewItem(new string[] { "003", "Nguyễn Hoàng
    Nam", "19", "Nam", "CTK42" });
    this.lvDanhSach.Items.Add(lvitem);
    lvitem = new ListViewItem("004");
    lvitem.SubItems.Add("Lê Thị Lan Hương");
    lvitem.SubItems.Add("22");
    lvitem.SubItems.Add("Nữ");
```

```

        lvitem.SubItems.Add("CTK40");
        lvDanhSach.Items.Add(lvitem);
    }
}

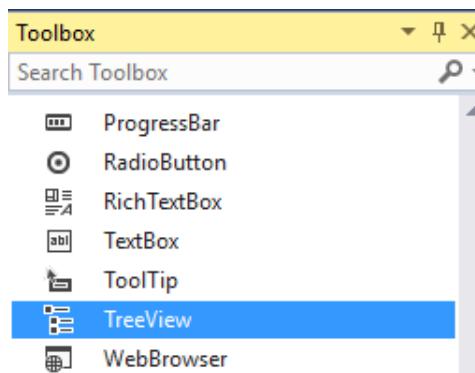
```

1.5.2. Điều khiển TreeView

TreeView là điều khiển dùng để bố trí danh sách các đối tượng theo dạng phân cấp có cấu trúc hình cây (Hình 1.53), mỗi đối tượng được biểu diễn là một Node thuộc lớp TreeNode. Mỗi một Node có thể chứa các Node khác. TreeView thường dùng sự kiện NodeMouseClick để xử lý khi người dùng click chọn một Node trên TreeView. Bảng 1.36 mô tả một số phương thức chính của TreeView. Ngoài ra, TreeView cũng có một số thuộc tính đặc trưng, được mô tả trong Bảng 1.37.

Bảng 1.36. Các phương thức thường dùng của TreeView

Phương thức	Mô tả
GetNodeCount()	Đếm số lượng Node trong TreeView
ExpandAll()	Hiển thị tất cả các Node trong TreeView
CollapseAll()	Thu gọn tất cả các Node trong TreeView



Hình 1.53. Điều khiển TreeView trên thanh Toolbox

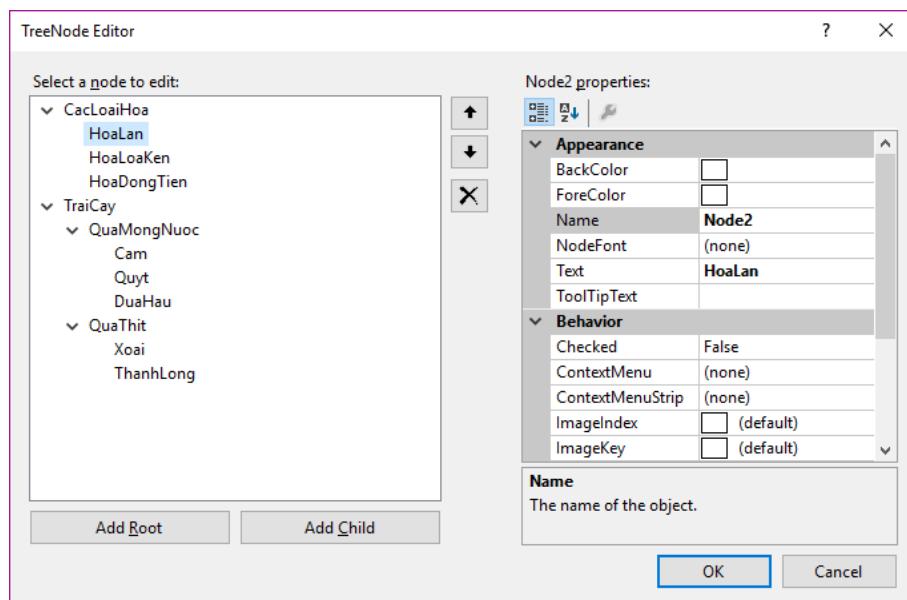
Bảng 1.37. Các thuộc tính thường dùng của TreeView

Thuộc tính	Mô tả
CheckBoxes	Nhận giá trị true hoặc false, cho phép hiển thị CheckBox trên TreeView hay không
ImageIndex	Nhận giá trị Index image của TreeView. Khi gán chỉ số cho thuộc tính ImageIndex thì hình hiển thị trước mỗi Node là hình có chỉ số tương ứng trong ImageList
ImageList	Nhận một ImageList cho TreeView, thiết lập hiển thị hình ảnh trước mỗi Node trong TreeView
Nodes	Trả về tập các đối tượng TreeNode của TreeView
PathSeparator	Thiết lập ký tự phân tách trong đường dẫn
SelectedNode	Trả về các Nodes đang được chọn trong TreeView
TopNode	Trả về TreeNode đầu tiên trong TreeView

Để thiết lập các Node cho TreeView, click chọn thuộc tính Nodes, cửa sổ TreeNode Editor hiển thị như trong Hình 1.54, để thêm một nút, chúng ta click chọn Add Root và đặt thuộc tính Text cho Node, để thêm Node con của một Node thì click Add Child. Để xóa một Node, click chọn biểu tượng chữ X trên thanh dọc ở giữa cửa sổ TreeNode Editor, khi xóa một Node, các Node con của nó cũng bị xóa theo. Ngoài ra, chúng ta có thể viết mã nguồn để khởi tạo TreeNode như sau:

```
TreeNode (string label);
TreeNode (string label, int imageIndex, int selectedImageIndex);
```

Tương tự như TreeView, TreeNode cũng có một số thuộc tính và phương thức được giới thiệu trong Bảng 1.38.



Hình 1.54. Cửa sổ TreeNode Editor

Bảng 1.38. Các thuộc tính và phương thức thường dùng của TreeNode

Thuộc tính/ Phương thức	Mô tả
FirstNode	Trả về Node con đầu tiên của Node hiện tại
Index	Trả về vị trí của Node hiện tại trong tập Node hoặc của TreeView
IsExpanded	Trả về kết quả liệu các Node con của Node hiện tại có đang được mở rộng hay không?
isSelected	Trả về kết quả Node có đang được chọn hay không?
Nodes	Lấy tập đối tượng TreeNode của Node hiện tại
Add()	Thêm một Node mới
Remove()	Xóa một Node trong TreeNode
Insert()	Chèn một Node vào TreeNode
Clear()	Xóa Node hiện tại cùng toàn bộ các Node con của nó

Ví dụ, để thiết kế Form như Hình 1.55, ta cần có một TreeView, một ImageList, sau khi đã thêm hình vào ImageList, sự kiện Load của Form được viết như sau:

```
private void frmTreeView_Load(object sender, EventArgs e)
{
    TreeNode rNode, cNode;
    rNode = this.treeViewThucVat.Nodes.Add("Các loại hoa");
    rNode.ImageIndex = 0;

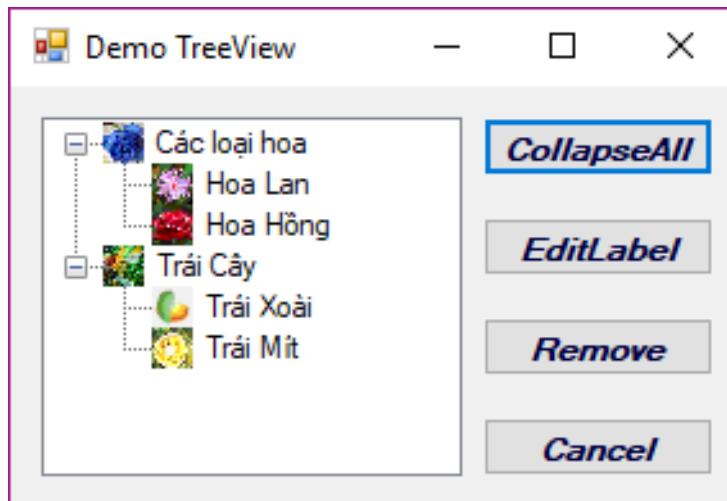
    cNode = new TreeNode("Hoa Lan");
    cNode.ImageIndex = 1;
    rNode.Nodes.Add(cNode);

    cNode = new TreeNode("Hoa Hồng", 2, 2);
    rNode.Nodes.Add(cNode);

    rNode = this.treeViewThucVat.Nodes.Add("Trái Cây");
    rNode.ImageIndex = 3;

    cNode = new TreeNode("Trái Xoài");
    cNode.ImageIndex = 4;
    rNode.Nodes.Add(cNode);

    cNode = new TreeNode("Trái Mít");
    cNode.ImageIndex = 5;
    rNode.Nodes.Add(cNode);
}
```

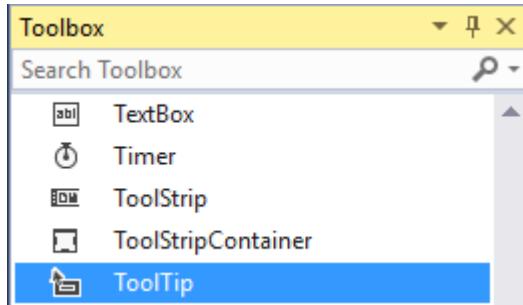


Hình 1.55. Minh họa TreeView

Các sự kiện của các nút bấm trên Form, độc giả tự viết bằng cách gọi các phương thức được mô tả trong Bảng 1.36.

1.5.3. Điều khiển ToolTip, ErrorProvider

ToolTip là điều khiển hiển thị các thông tin chú thích hoặc hướng dẫn khi người dùng rê chuột vào điều khiển có thiết lập ToolTip (Hình 1.56). ToolTip có một số thuộc tính và phương thức được mô tả trong Bảng 1.39.

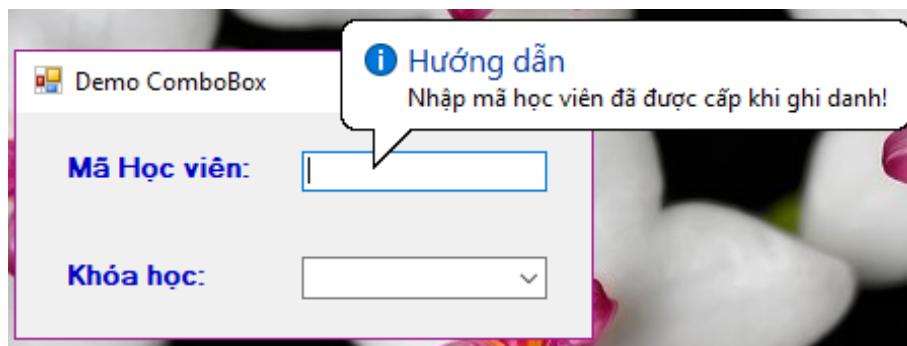


Hình 1.56. Điều khiển ToolTip trên thanh Toolbox

Bảng 1.39. Các thuộc tính và phương thức cơ bản của ToolTip

Thuộc tính/ Phương thức	Mô tả
IsBalloon	Hiển thị ToolTip ở dạng bo góc
ToolTipIcon	Biểu tượng xuất hiện trên cửa sổ ToolTip
ToolTipTitle	Tiêu đề của cửa sổ ToolTip
SetToolTip()	Khai báo chuỗi xuất hiện trên cửa sổ ToolTip
GetToolTip()	Trả về nội dung chuỗi trên cửa sổ ToolTip
RemoveAll()	Loại bỏ tất cả các ToolTip của tất cả các điều khiển

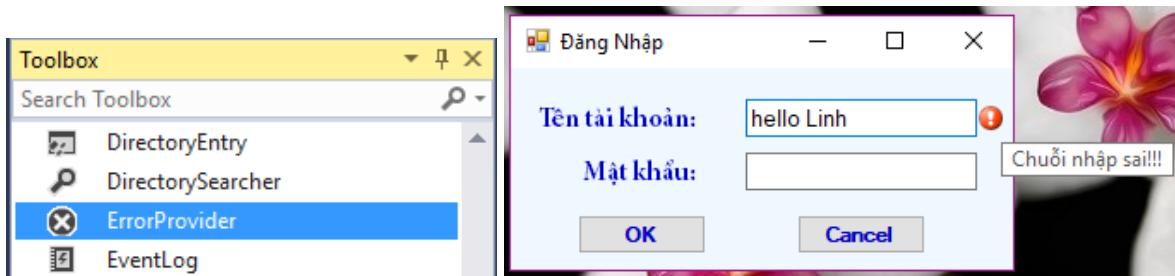
Để sử dụng ToolTip cho một điều khiển nào đó, ta kéo điều khiển ToolTip từ ToolBox vào Form, đặt tên và thiết lập các thuộc tính cần thiết, sau đó, click chọn điều khiển muốn gán ToolTip sẽ thấy có thuộc tính ToolTip on <tên ToolTip đã đặt>, thiết lập đoạn văn bản muốn hiển thị lên ToolTip trong thuộc tính này là xong (Hình 1.57).



Hình 1.57. Minh họa sử dụng ToolTip

Điều khiển ErrorProvider giúp thông báo cho người dùng lỗi nhập liệu trên Form (Hình 1.58). Thông thường, khi dữ liệu người dùng nhập bị lỗi ErrorProvider sẽ hiển thị một icon Error bên cạnh điều khiển để thông báo lỗi. Cách sử dụng ErrorProvider tương

tự như ToolTip, khi đó ta thiết lập chuỗi hiển thị trên ErrorProvider của điều khiển cho thuộc tính Error on <Tên ErrorProvider>. Điều khiển này cũng có một số thuộc tính và phương thức cơ bản như trong Bảng 1.40.



Hình 1.58. Điều khiển ErrorProvider trên thanh Toolbox và minh họa

Bảng 1.40. Các thuộc tính và phương thức thường dùng của ErrorProvider

Thuộc tính/ Phương thức	Mô tả
Icon	Biểu tượng xuất hiện bên cạnh thông báo lỗi
BlinkRate	Tốc độ nháy nháy, tính theo mili giây
BlinkStyle	Kiểu nháy nháy
SetError()	Thiết lập chuỗi thông báo lỗi của điều khiển
GetError()	Trả về chuỗi thông báo lỗi của điều khiển
Clear()	Loại bỏ tất cả các thông báo lỗi của tất cả các điều khiển trên Form

1.5.4. Điều khiển UserControl

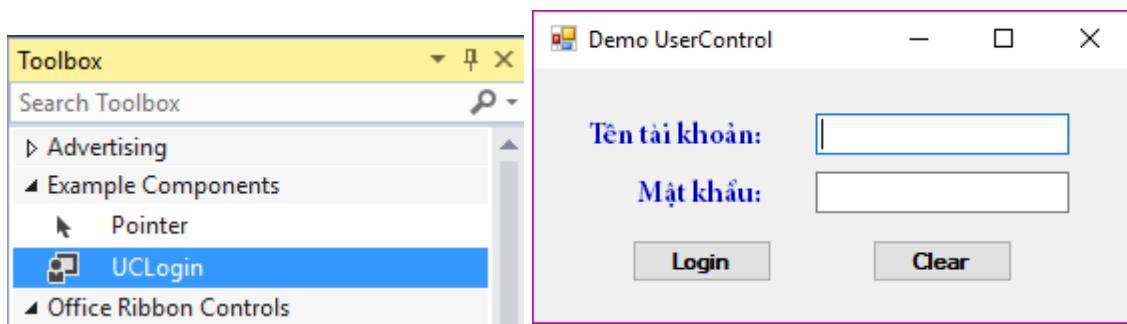
UserControl là dạng điều khiển do lập trình viên tự tạo ra. Trên thực tế, việc xây dựng một UserControl cũng giống như xây dựng Form, lập trình viên sẽ tự tay thiết kế Form với một số điều khiển chứa trong đó, để có thể tái sử dụng nhiều lần, lập trình viên sẽ đưa Form đã thiết kế về dạng UserControl.

Để bắt đầu tạo UserControl, click chọn menu Project chọn Add User Control... Hoặc click chuột phải lên tên của dự án đang mở trong Solution Explorer chọn Add chọn User Control... Ví dụ tạo UserControl đặt tên là UCLogin, thiết kế như Hình 1.59.



Hình 1.59. Minh họa thiết kế UserControl

Tiến hành chạy dự án (hoặc chạy lệnh Build), UserControl sẽ được nhúng thêm vào thanh Toolbox. Để sử dụng UserControl vừa thiết kế, chúng ta tạo một Form mới và kéo điều khiển này từ ToolBox vào giao diện Form (Hình 1.60).

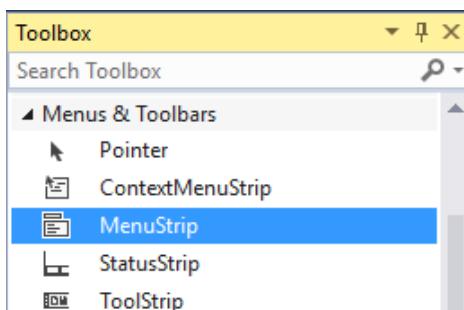


Hình 1.60. Kết quả sử dụng UserControl

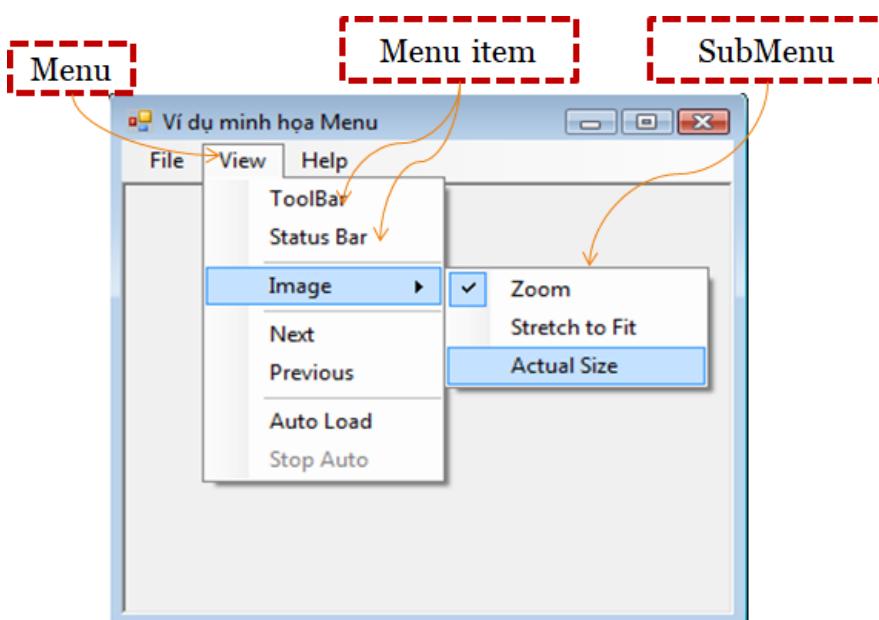
1.6. Các điều khiển kiểu Menu

1.6.1. ToolStrip

ToolStrip là điều khiển cung cấp các nhóm lệnh có liên quan đến nhau dưới dạng một hệ thống thực đơn hiển thị trên Form. Menu có thể có một hoặc nhiều cấp, mỗi Menu Item cấp đầu tiên là một ToolStripMenuItem. ToolStrip được đặt trong nhóm Menus & ToolBars của ToolBox (Hình 1.61). Hình 1.62 giải thích các thành phần của một hệ thống Menu.

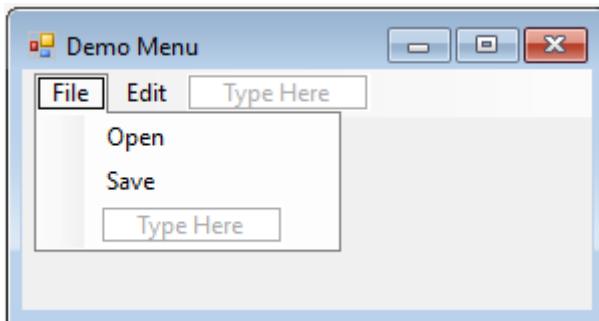


Hình 1.61. Điều khiển ToolStrip trong Toolbox



Hình 1.62. Các thành phần của Menu

Để tạo một Menu, chúng ta kéo điều khiển ToolStrip từ ToolBox vào Form, thanhMenuBar sẽ xuất hiện ở vị trí trên cùng của Form, tại đây ta tiến hành tạo các MenuItem cho Menu (Hình 1.63). MenuItem có thể là Menu, TextBox, ComboBox hoặc là một đường thẳng nằm ngang. Bảng 1.41 và Bảng 1.42 là một số thuộc tính chính của ToolStripMenuItem và ToolStripMenuItem.



Hình 1.63. Minh họa tạo Menu trên Form

Bảng 1.41. Các thuộc tính thường dùng của ToolStrip

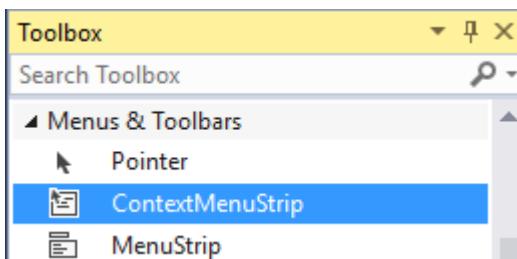
Thuộc tính	Mô tả
Items	Chứa các ToolStripMenuItem, là các Menu ở cấp đầu tiên
RightToLeft	Nhận giá trị Yes hoặc No, nếu là Yes thì sẽ hiển thị Menu từ phải qua trái
TextDirection	Hướng của chữ trong Menu

Bảng 1.42. Các thuộc tính thường dùng của ToolStripMenuItem

Thuộc tính	Mô tả
Checked	Xác định trạng thái check của Menu Item
Index	Chỉ mục của Menu Item trong Menu cha
DropDownItems	Chứa những Menu Item con
ShortcutKeys	Thiết lập phím tắt cho Menu Item

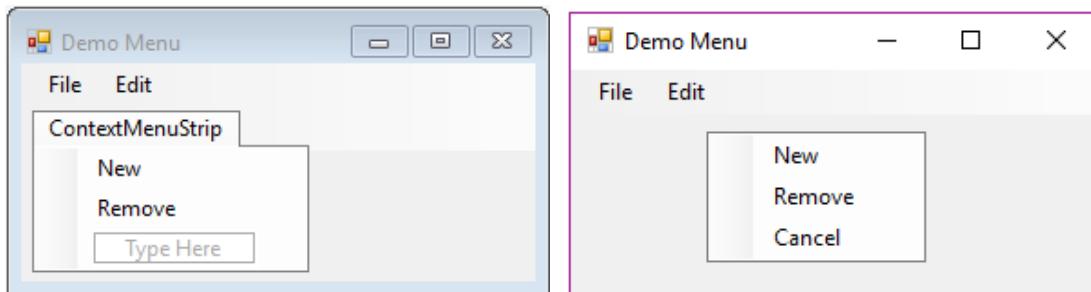
1.6.2. ContextMenuStrip

ContextMenuStrip dùng để thiết kế Menu ngữ cảnh dạng Popup, nghĩa là khi người dùng click chuột phải lên điều khiển nào thì sẽ hiển thị ra ContextMenuStrip đã thiết lập cho điều khiển tương ứng (Hình 1.64).



Hình 1.64. Điều khiển ContextMenuStrip trong ToolBox

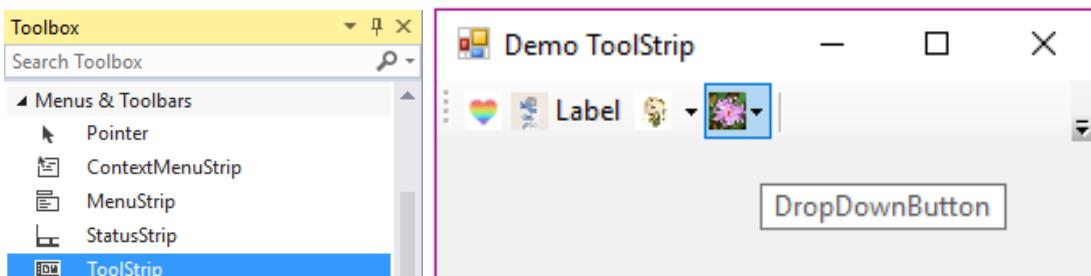
Để tạo ContextMenuStrip, ta kéo điều khiển ContextMenuStrip từ ToolBox vào Form và thiết lập các Menu Item cho nó. Muốn sử dụng ContextMenuStrip cho điều khiển nào ta chỉ cần thiết lập thuộc tính ContextMenuStrip là tên của một điều khiển ContextMenuStrip (Hình 1.65).



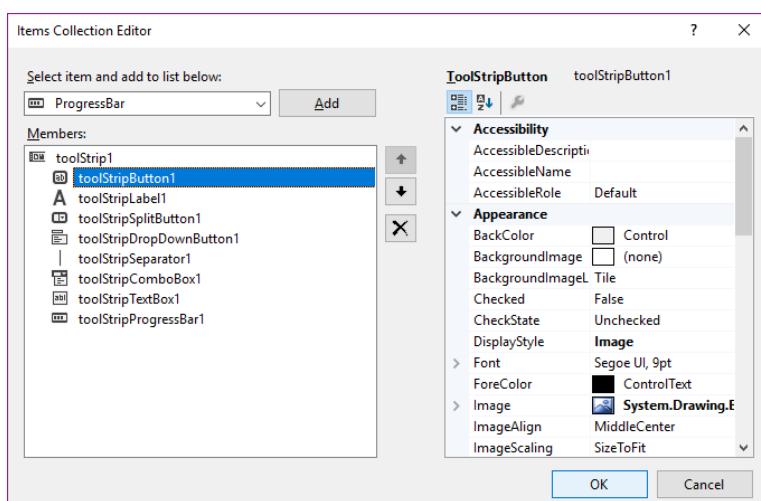
Hình 1.65. Thiết lập ContextMenuStrip và minh họa hiển thị lên Form

1.6.3. ToolStrip

Điều khiển ToolStrip là điều khiển có thể chứa nhiều điều khiển khác, cho phép tạo thanh công cụToolBar trên các ứng dụng Windows Forms (Hình 1.66). ToolStrip có thuộc tính Items là một tập hợp các điều khiển chứa trong ToolStrip. Để thêm một điều khiển vào ToolStrip, click chọn thuộc tính Items, cửa sổ Items Collection Editor hiện ra, ta chọn loại điều khiển muốn thêm vào và bấm Add (Hình 1.67). ToolStrip sử dụng sự kiện Click để xử lý khi người dùng click chọn một Item trong ToolStrip.



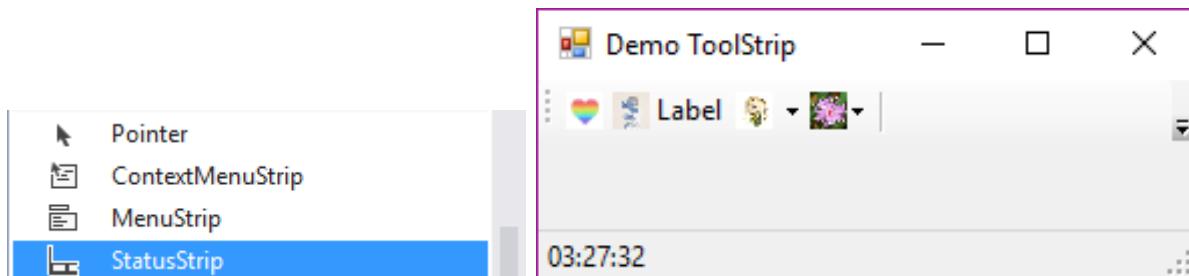
Hình 1.66. Điều khiển ToolStrip trên thanh ToolBox và minh họa



Hình 1.67. Cửa sổ Items Collection Editor của ToolStrip

1.6.4. StatusStrip

Điều khiển StatusStrip dùng để hiển thị một số thông tin trạng thái của ứng dụng (Hình 1.68). Điều khiển này thường nằm ở dưới cùng trong giao diện Form. StatusStrip cũng là một điều khiển dạng Container, các điều khiển mà nó có thể chứa đó là:StatusLabel, ProgressBar, DropDownButton và SplitButton. Điều khiển được sử dụng nhiều nhất trong StatusStrip là StatusLabel.

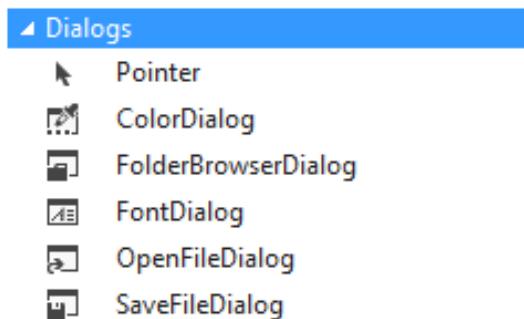


Hình 1.68. StatusStrip trên thanh ToolBox và minh họa

Ví dụ: Để hiển thị thời gian hiện tại lên StatusStrip của Form chúng ta kéo điều khiển StatusStrip vào Form, sau đó kéo điều khiển Timer vào. Thiết lập sự kiện Tick cho Timer để biểu thị sự thay đổi thời gian trên StatusStrip.

1.7. Các dạng hộp thoại (Dialog)

Các điều khiển dạng Dialog là điều khiển dạng hiển thị một hộp thoại để cho phép người dùng thao tác trên đó. Các điều khiển Dialog được đặt trong nhóm Dialogs của ToolBox (Hình 1.69).



Hình 1.69. Nhóm điều khiển Dialogs

1.7.1. OpenFileDialog và SaveFileDialog

Điều khiển OpenFileDialog hoặc SaveFileDialog hiển thị một hộp thoại cho phép người dùng mở hoặc lưu một (một vài) tập tin trên bộ nhớ của máy. Cả hai dạng hộp thoại này có cùng một số thuộc tính được mô tả trong Bảng 1.43, riêng OpenFileDialog có thêm một số thuộc tính khác (Bảng 1.44).

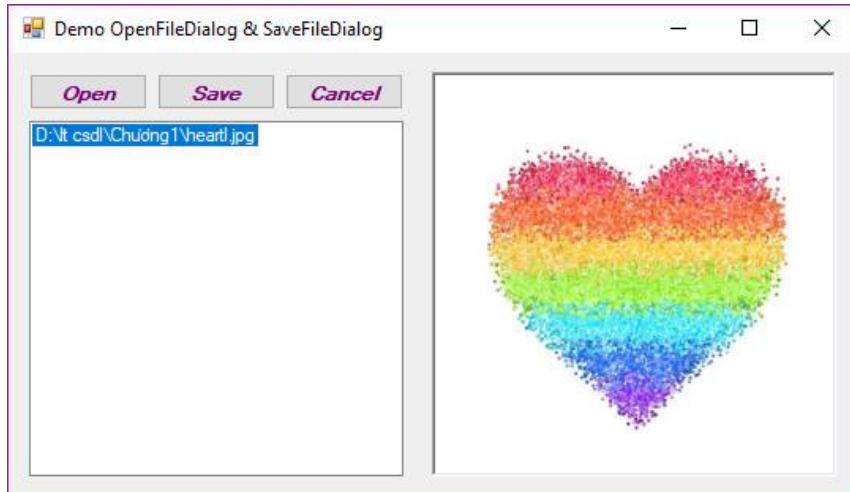
Bảng 1.43. Các thuộc tính thường dùng của OpenFileDialog và SaveFileDialog

Thuộc tính	Mô tả
FileName	Trả về tên một tập tin chọn khi nhấn OK
FileNames	Trả về tên nhiều tập tin chọn khi nhấn OK
Filter	Thuộc tính này dùng để quy định các loại tập tin sẽ được hiển thị trong hộp thoại (hình ảnh, văn bản,...)
InitialDirectory	Quy định thư mục mặc định khi mở hộp thoại
RestoreDirectory	Mỗi khi mở hộp thoại Open và Save, thư mục được người dùng chọn lần cuối trước đó sẽ được hiển thị.

Bảng 1.44. Thuộc tính thường dùng của OpenFileDialog

Thuộc tính	Mô tả
MultiSelect	Nhận giá trị true hoặc false: Có cho phép chọn nhiều tập tin một lúc hay không
ShowReadOnly	Nhận giá trị true hoặc false: Cho phép hiển thị CheckBox “Open As Read Only” trong hộp thoại Open hay không.
ReadOnlyChecked	Nhận giá trị true hoặc false: Quy định giá trị của CheckBox là ReadOnlyChecked có được chọn hay không.

Ví dụ: Thiết kế giao diện Form như Hình 1.70 cho phép mở và xem tập tin hình ảnh trên Form.



Hình 1.70. Ví dụ OpenFileDialog và SaveFileDialog

Trước hết, tạo sự kiện Click cho nút Open như sau:

```
private void btnOpen_Click(object sender, EventArgs e)
{
    openFfileDlg.Title = "Open Dialog";
    openFfileDlg.Multiselect = true;
    openFfileDlg.Filter = "Image Files (JPEG, GIF, BMP, etc.)|"
        + "*.jpg;*.jpeg;*.gif;*.bmp;*.tif;*.tiff;*.png|"
        + "JPEG files (*.jpg;*.jpeg)|*.jpg;*.jpeg|"
```

```
+ "GIF files (*.gif)|*.gif| BMP files  
(*.bmp)|*.bmp|"  
+ "TIFF files (*.tif;*.tiff)|*.tif;*.tiff|"  
+ "PNG files (*.png)|*.png| All files (*.*)|*.*";  
if (openFileDlg.ShowDialog() == DialogResult.OK)  
{  
    string[] filenames = openFileDlg.FileNames;  
    for (int i = 0; i < filenames.Length; i++)  
        this.lvListFile.Items.Add(new ListViewItem(filenames[i]));  
}  
}  
}
```

Tiếp theo, tạo sự kiện Click cho nút Save:

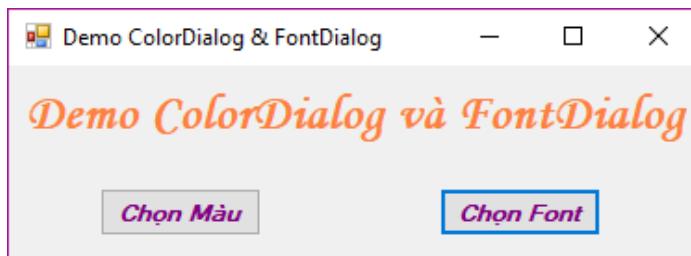
```
private void btnSave_Click(object sender, EventArgs e)  
{  
    saveFileDlg.Title = "Save file ..";  
    saveFileDlg.Filter = "Image Files (JPEG,BMP,GIF,...)|(*.jpeg;  
*.jpg;)|"  
                     +"Bitmap files (*.bmp)|*.bmp| All files  
(*.*);*.*";  
if (saveFileDlg.ShowDialog() == DialogResult.OK)  
{  
    try {  
        Image im = pbImage.Image;  
        im.Save(saveFileDlg.FileName, ImageFormat.Bmp);  
    }  
    catch {}  
}  
}
```

Cuối cùng, tạo sự kiện SelectedIndexChanged cho ListView để khi người dùng chọn, ảnh sẽ hiện lên trên PictureBox có tên là pbImage.

```
private void lvListFile_SelectedIndexChanged(object sender, EventArgs e)  
{  
    int i = lvListFile.SelectedItems.Count - 1;  
    if (i >= 0)  
    {  
        ListViewItem lvitem = lvListFile.SelectedItems[0];  
        this.pbImage.ImageLocation = lvitem.Text;  
    }  
}
```

1.7.2. ColorDialog và FontDialog

Điều khiển ColorDialog hiển thị hộp thoại màu cho phép người dùng thay đổi màu cho các điều khiển. Điều khiển FontDialog hiển thị hộp thoại Fonts cho phép người dùng thay đổi font chữ, kiểu chữ, kích thước chữ và cả màu chữ cho các điều khiển (Hình 1.71). Bảng 1.45 mô tả một số thuộc tính của hộp thoại ColorDialog và bảng 1.46 mô tả một số thuộc tính của hộp thoại FontDialog.



Hình 1.71. Minh họa ColorDialog và FontDialog

Bảng 1.45. Các thuộc tính thường dùng của ColorDialog

Thuộc tính	Mô tả
AllowFullOpen	Cho phép hiển thị đầy đủ hộp thoại màu hay không
FullOpen	Cho phép hiển thị hộp thoại đầy đủ, bao gồm cả Custom color hay không
Color	Get hoặc set giá trị màu cho hộp thoại

Bảng 1.46. Các thuộc tính thường dùng của FontDialog

Thuộc tính	Mô tả
Color	Thiết lập giá trị màu trên hộp thoại
Font	Thiết lập giá trị font của hộp thoại
ShowColor	Hiển thị ComboBox cho phép lựa chọn màu sắc hay không
ShowEffects	Hiển thị lựa chọn hiệu ứng chữ: StrileOut, Underline hay không
ShowApply	Hiển thị nút Apply trong hộp thoại hay không

1.8. Xử lý sự kiện chuột, bàn phím

Sự kiện được xem như kết quả của một hành động nào đó, ví dụ như khi đang lái xe, bạn đạp thắng (hành động) thì xe sẽ dừng lại (sự kiện). Windows Forms cũng có các sự kiện, nó sử dụng sự kiện để xử lý các thao tác bất kỳ của người dùng trên chương trình. Trong lập trình Windows Forms, chuột và bàn phím được xem như là hai hình thức đầu vào duy nhất của người dùng, chính vì vậy việc xử lý các hành động của chuột và bàn phím cũng là một trong những loại sự kiện quan trọng nhất.

1.8.1. Xử lý sự kiện chuột

Các sự kiện chuột xảy ra khi người dùng tương tác với điều khiển thông qua chuột, ví dụ như nhấp chuột hay di chuyển chuột. Các sự kiện chuột có thể được xử lý cho mọi điều khiển xuất phát từ lớp System.Windows.Forms.Control. Đối với hầu hết các sự kiện chuột, thông tin về sự kiện được chuyển đến phương thức xử lý sự kiện thông qua một đối tượng của lớp MouseEventArgs và sử dụng ủy nhiệm hàm (delegate) MouseEventHandler để tạo trình xử lý chuột. Mỗi phương thức xử lý sự kiện chuột sử dụng đối số là một đối tượng MouseEventArgs (Bảng 1.47, Bảng 1.48).

Lớp MouseEventArgs bao gồm các thông tin liên quan đến sự kiện chuột, như tọa độ (x,y) của con trỏ chuột, nút chuột được nhấn (phải, trái, hoặc giữa) và số lần click

chuột. Các tọa độ x và y của đối tượng MouseEventArgs là tọa độ tương quan trong điều khiển xảy ra sự kiện. Tức là, điểm (0,0) đại diện cho góc trên bên trái của điều khiển nơi diễn ra sự kiện chuột (Bảng 1.49).

Bảng 1.47. Một số sự kiện chuột với tham số kiểu EventArgs

Sự kiện	Mô tả
MouseEnter	Xảy ra khi chuột trỏ vào vùng biên của điều khiển
MouseHover	Xảy ra khi con trỏ chuột di chuyển trong vùng biên của điều khiển
MouseLeave	Xảy ra khi con trỏ rời khỏi vùng biên của điều khiển

Bảng 1.48. Sự kiện chuột với tham số kiểu MouseEventArgs

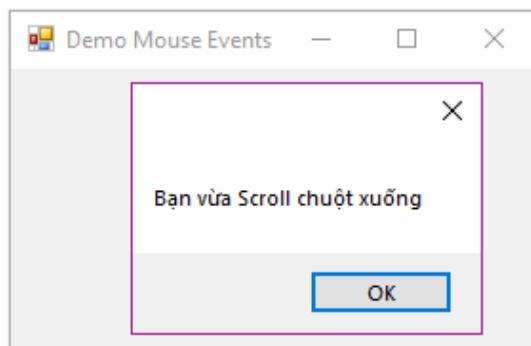
Sự kiện	Mô tả
MouseDown/MouseUp	Xảy ra khi nút trái hoặc phải chuột được nhấn/thả khi con trỏ đang ở trong vùng biên của điều khiển
MouseMove	Xảy ra khi con trỏ chuột di chuyển trong vùng biên của điều khiển
MouseWheel	Xảy ra khi di chuyển con lăn của chuột lúc điều khiển đang nhận focus

Bảng 1.49. Các thuộc tính của lớp MouseEventArgs

Thuộc tính	Mô tả
Button	Xác định nút chuột nào được nhấn {Left, Right, Middle, None}, có kiểu MouseButtons
Clicks	Số lần nút chuột được nhấn
x	Toạ độ x của con trỏ chuột trong điều khiển
y	Toạ độ y của con trỏ chuột trong điều khiển

Ví dụ: Xử lý sự kiện MouseWheel của chuột (Hình 1.72) bằng cách viết phương thức ghi đè (override) sự kiện OnMouseWheel như sau:

```
protected override void OnMouseWheel(MouseEventArgs e)
{
    if (e.Delta > 0) MessageBox.Show("Bạn vừa Scroll chuột lên");
    else      MessageBox.Show("Bạn vừa Scroll chuột xuống");
}
```



Hình 1.72. Minh họa xử lý sự kiện chuột

1.8.2. Xử lý sự kiện bàn phím

Các sự kiện bàn phím xảy ra khi người dùng nhấn hoặc nhả một phím hay tổ hợp phím trên bàn phím. Cũng giống như sự kiện chuột, sự kiện bàn phím có thể được xử lý cho bất kỳ điều khiển nào kể thừa từ System.Windows.Forms.Control.

Có ba loại sự kiện bàn phím, đó là: KeyPress, KeyUp và KeyDown. Sự kiện KeyPress xảy ra khi người dùng nhấn phím ký tự hoặc phím Space hoặc phím Backspace. Phím cụ thể có thể được xác định bằng thuộc tính KeyChar của trình xử lý sự kiện thông qua tham số KeyPressEventEventArgs. Sự kiện KeyPress không cho biết thông tin liệu các phím hỗ trợ (ví dụ: Shift, Alt và Ctrl) có được nhấn khi xảy ra sự kiện phím hay không. Các thông tin này có thể được cung cấp thông qua các sự kiện KeyUp hoặc KeyDown. Một số thuộc tính của lớp KeyPressEventEventArgs và lớp KeyEventArgs được mô tả trong Bảng 1.50 và Bảng 1.51, trong khi một số sự kiện lại được trình bày trong Bảng 1.52.

Bảng 1.50. Các thuộc tính của lớp KeyPressEventEventArgs

Thuộc tính	Mô tả
KeyChar	Trả về mã ASCII của phím được nhấn
Handled	Cho biết sự kiện KeyPress đã được xử lý hay chưa

Bảng 1.51. Các thuộc tính của lớp KeyEventArgs

Thuộc tính	Mô tả
Alt/Control/Shift	Trả về trạng thái của các phím hỗ trợ
Handled	Cho biết sự kiện đã được xử lý hay chưa
KeyCode	Trả về mã ký tự được định nghĩa trong <i>Keys enumeration</i>
KeyData	Trả về mã ký tự cùng với thông tin phím hỗ trợ
KeyValue	Trả về một số kiểu int, đây chính là mã <i>Windows Virtual Key Code</i>
Modifier	Trả về giá trị của phím hỗ trợ

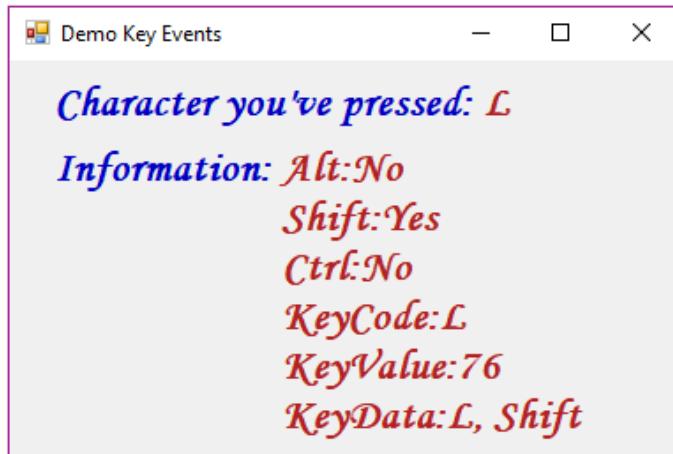
Bảng 1.52. Sự kiện phím với tham số kiểu KeyEventArgs

Sự kiện	Mô tả
KeyDown	Xảy ra khi phím được nhấn
KeyUp	Xảy ra khi phím được nhả

Ví dụ, viết chương trình hiển thị phím và các thông tin phím hỗ trợ mà người dùng bấm trên bàn phím (Hình 1.73), trước tiên ta lấy thông tin phím bấm, sau đó lấy thông tin phím hỗ trợ. Hai thao tác trên được viết trong các sự kiện sau:

```
private void frmKeyEvent_KeyPress(object sender, KeyPressEventEventArgs e)
{
    this.lblChar.Text = e.KeyChar.ToString();
}
```

```
private void frmKeyEvent_KeyDown(object sender, KeyEventArgs e)
{
    this.lblInfo.Text = "Alt:" + (e.Alt ? "Yes" : "No") + '\n' +
        "Shift:" + (e.Shift ? "Yes" : "No") + '\n' +
        "Ctrl:" + (e.Control ? "Yes" : "No") + '\n' +
        "KeyCode:" + e.KeyCode + "\n" + "KeyValue:" + e.KeyValue + "\n" +
        "KeyData:" + e.KeyData;
}
```



Hình 1.73. Ví dụ xử lý sự kiện bàn phím

1.9. Kết chương

Chương này đã mang đến cho người đọc cái nhìn tổng quan về lập trình giao diện, cụ thể là ứng dụng Windows Form trong bộ công cụ Visual Studio .NET, cung cấp cho người đọc những kiến thức cơ bản về các điều khiển, sự kiện thường dùng trong Windows Application, đi kèm với đó là những ví dụ minh họa đơn giản và cụ thể giúp cho người đọc có thể đọc hiểu và thực hành một cách dễ dàng.

Bài tập:

Thiết kế Form như Hình 1.73. Chương trình cho phép người dùng nhập vào các điều khiển như trong GroupBox Thông tin. Khi người dùng click vào một Item trong ListView thì hiển thị các thông tin tương ứng của Item lên GroupBox Thông tin. Chức năng các nút như sau:

- Thêm: Lấy thông tin người dùng nhập trong GroupBox Thông tin chèn vào ListView;
- Xóa: Khi người dùng chọn item trong ListView rồi bấm nút Xóa thì xóa Item khỏi ListView;
- Sửa: Cập nhật thông tin đã sửa đổi của Item vào ListView;
- Mặc định: Xóa trống các trường thông tin trong GroupBox Thông tin;
- Thoát: Đóng chương trình.

Giáo trình Lập trình Cơ sở dữ liệu

Hình 1.73. Form của Bài tập 1

CHƯƠNG 2. BIỂU DIỄN DỮ LIỆU BÁN CẤU TRÚC

Khi xây dựng một chương trình, thường thì chương trình đó phải kết nối với cơ sở dữ liệu để đọc và truy vấn dữ liệu để xử lý. Có ba loại dữ liệu được sử dụng là dữ liệu phi cấu trúc (Unstructured Data), dữ liệu có cấu trúc (Structured Data), và dữ liệu bán cấu trúc (Semi-Structured Data). Dữ liệu phi cấu trúc là các dữ liệu được phát sinh trong quá trình hoạt động của con người (ví dụ: Tập tin, văn bản, các cuộc gọi điện,...); Dữ liệu có cấu trúc là dạng dữ liệu được lưu trữ theo một cấu trúc nhất định, biểu diễn theo các mẫu tin (record) và trường (field) (ví dụ: Dữ liệu được lưu trong các hệ quản trị cơ sở dữ liệu SQL Server, Oracle, MySQL...); Dữ liệu bán cấu trúc là dạng dữ liệu mà thông tin không nằm trong một hệ quản trị cơ sở dữ liệu nhưng lại có các cấu trúc và thuộc tính để tổ chức và phân biệt với nhau (ví dụ: Email là một dạng dữ liệu có các trường như: Người gửi, Chủ đề, Nội dung...). Các loại dữ liệu này có thể được biến đổi để chuyển từ loại này sang loại khác và chúng cũng có mối quan hệ tương hỗ với nhau.

Đối với các chương trình có kết nối với cơ sở dữ liệu, người ta thường kết nối với cơ sở dữ liệu dạng có cấu trúc, việc này giúp thao tác thuận tiện và dễ quản lý chương trình. Tuy vậy, với sự phát triển của công nghệ thông tin, tùy vào mức độ ứng dụng mà một số chương trình có thể sử dụng dữ liệu dạng bán cấu trúc để lưu trữ và xử lý dữ liệu. Chương này giúp người đọc tìm hiểu một số kiểu dữ liệu bán cấu trúc thông dụng đang được sử dụng cho các ứng dụng hiện nay.

Những nội dung chính sẽ được trình bày bao gồm:

- Giới thiệu về dữ liệu bán cấu trúc;
- Kiểu dữ liệu XML;
- Kiểu dữ liệu JSON;
- Kiểu dữ liệu BSON.

2.1. Dữ liệu bán cấu trúc

Khi xây dựng một ứng dụng thì dữ liệu thường được lưu trữ ở một hệ quản trị cơ sở dữ liệu nào đó ví dụ như SQL Server, MySQL, Oracle.... Những hệ quản trị cơ sở dữ liệu này dùng để lưu dạng dữ liệu có cấu trúc và phải cài đặt các phần mềm tương ứng để thao tác và sử dụng. Nếu giả sử chương trình cần xây dựng một dạng ứng dụng nhỏ, gọn và đơn giản thì sao? Nếu phải sử dụng các hệ quản trị như trên sẽ phải cài đặt nhiều thứ sẽ gây phiền hà cho người dùng. Một ví dụ khác là trong một chương trình, thỉnh thoảng có những dữ liệu rất đơn giản nhưng cần phải lưu trữ (ví dụ: chương trình quản lý cho một nhà hàng có thông tin về địa chỉ, số điện thoại... của nhà hàng đó, dữ liệu này là đơn giản và không cần tổ chức thành 1 bảng). Lúc này, người ta sẽ chọn một giải pháp nào đó để lưu trữ mà không dùng đến hệ quản trị cơ sở dữ liệu, lúc đó dữ liệu dạng bán cấu trúc ra đời.

Dữ liệu bán cấu trúc là dạng dữ liệu mà thông tin không được tổ chức và lưu trữ trong hệ quản trị cơ sở dữ liệu nhưng lại có các thuộc tính để quy định cách tổ chức và

phân tích dữ liệu một cách dễ dàng. Nói cách khác, dữ liệu bán cấu trúc không lộn xộn và không được kiểm soát như dữ liệu phi cấu trúc nhưng không cứng nhắc và dễ định lượng như dữ liệu có cấu trúc. Một ví dụ điển hình về dữ liệu bán cấu trúc là việc sử dụng các thẻ HTML để lưu trữ nội dung văn bản Web (Hình 2.1). Trong đó, dữ liệu là 2 dòng trong thẻ `<h1>` và `<p>`, 2 dòng này có quy tắc là phải nằm trong thẻ nào, ở vị trí nào để trình duyệt có thể đọc được.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Hình 2.1. Các thẻ HTML biểu diễn dữ liệu

Nguồn: <https://www.w3schools.com/>

Bảng 2.1 minh họa một số loại dữ liệu dạng bán cấu trúc được sử dụng thông dụng trong các ngôn ngữ lập trình. Khi dữ liệu được lưu trong các kiểu này, dữ liệu sẽ rất nhẹ nhưng vẫn đảm bảo có tính cấu trúc để quản lý và thao tác trên đó.

Bảng 2.1. Một số loại hình dữ liệu bán cấu trúc

Kiểu dữ liệu	Ví dụ	Mô tả
Kiểu dữ liệu dạng tập tin	Là dạng dữ liệu được tạo ra bởi các tập tin, mỗi tập tin chứa nhiều mẫu tin là dòng dữ liệu, mỗi dòng có thể có các cột.	CSV, Excel, Foxpro, File Text...
Kiểu dữ liệu dạng thẻ	Kiểu dữ liệu cũng dạng tập tin nhưng trong đó chứa các thẻ để chứa dữ liệu, mỗi thẻ có thể có các thuộc tính để phân biệt.	XML, XAML, AIML, HTML...
Kiểu dữ liệu dạng trường cấu trúc	Là các kiểu dữ liệu có các trường để chứa thông tin, các trường này do người dùng định nghĩa.	JSON, BSON...
Kiểu dữ liệu dạng cấu trúc quy ước	Là dạng dữ liệu có các trường được tạo ra theo một quy định cho trước. Khi sử dụng kiểu này, người dùng phải lưu thông tin theo các trường có sẵn.	Email, RDF...

Các mục tiếp theo trong chương này sẽ nêu một số kiểu dữ liệu thông dụng thường được sử dụng để lưu trữ dữ liệu đơn giản trong quá trình lập trình. Các kiểu dữ liệu đó là XML, JSON, BSON. Ứng dụng thường xuyên của các kiểu dữ liệu này là dùng để xây dựng các dịch vụ (*service*) và API (*Application Programming Interface*) để tương tác giữa các hệ thống khác nhau.

2.2. Kiểu dữ liệu XML

XML (eXtensible Markup Language) hay còn gọi là ngôn ngữ đánh dấu mở rộng là một chuẩn ngôn ngữ do W3C (World Wide Web Consortium¹) đề xuất nhằm mục đích tạo ra một ngôn ngữ đánh dấu bởi các thẻ và có thể mô tả được nhiều kiểu dữ liệu khác nhau. Các đặc tả dữ liệu trong XML đều tuân thủ theo quy tắc và cú pháp của ngôn ngữ này. Không như ngôn ngữ HTML là tên các thẻ được quy định trước, XML cho phép người dùng tự định nghĩa thẻ để lưu dữ liệu. Các thẻ được định nghĩa tuân thủ theo cấu trúc cây với các nút lồng nhau.

2.2.1. Cú pháp của XML

Để biên tập XML, có thể dùng các công cụ như: iTaxViewer, Notepad... hoặc có thể sử dụng Visual Studio để biên tập. Tập tin XML có phần mở rộng là *.xml, trong đó được quy định bởi các nút, mỗi nút có thẻ đóng và thẻ mở, tuân theo cú pháp như sau:

```
<tennut>nội dung</tennut>
```

Trong đó: <tennut> là thẻ mở của nút; </tennut> là thẻ đóng của nút, phần ở giữa là nội dung dữ liệu cần chứa. Tên nút do người dùng tự định nghĩa, theo quy tắc không có khoảng trắng, phải bắt đầu bằng chữ cái.

Chương trình 2.1 sau đây là một đoạn XML minh họa quá trình lưu dữ liệu về nhà hàng như: Tên nhà hàng, địa chỉ, điện thoại, website. Vì dữ liệu này đơn giản, chỉ là một mẫu tin nên không cần đưa vào bảng cơ sở dữ liệu. Dữ liệu này được lưu trên XML và có thể đọc lên chương trình bằng ngôn ngữ lập trình (xem Chương 5 giáo trình này).

Chương trình 2.1. Ví dụ minh họa XML lưu trữ thông tin nhà hàng

```
<?xml version="1.0" encoding="utf-8" ?>
<nhahang>
    <tentiengViet>
        Nhà hàng Hoàng Tâm
    </tentiengViet>
    <tentiengAnh>
        Hoang Tam Restaurant
    </tentiengAnh>
    <diachi>
        221 Bà Triệu, Phường 4, TP. Đà Lạt
    </diachi>
    <dienthoai>
        02633 555 247
    </dienthoai>
    <website>
        www.hoangtamdl.vn
    </website>
</nhahang>
```

¹<https://www.w3.org/>

Trong đoạn chương trình trên, dòng đầu tiên được gọi là chỉ thị, đây là một dòng bắt buộc khi tạo một tập tin XML (trong Visual Studio là dòng được tạo ra mặc định). Sau đó là các nút lồng nhau, trong đó nút cha (root) là nút ngoài cùng, sau đó tới các nút con và nút cháu. Mỗi khi tạo ra một nút nào phải có thẻ mở và thẻ đóng cho nút đó.

2.2.2. Thuộc tính trong XML

Trong XML, ngoài các thẻ biểu diễn dữ liệu còn có các thuộc tính đi kèm thẻ để cung cấp thêm thông tin về phần tử đó. Thuộc tính của XML luôn có dạng cặp name-value và được đi kèm với thẻ. Một thuộc tính của XML có cú pháp như sau:

```
<elementname attribute1="value1" attribute2="value2">
    <!-- nội dung -->
</elementname>
```

Ở đây attribute1 và attribute2 chính là thuộc tính của thẻ elementname, value là giá trị của thuộc tính luôn nằm trong dấu ngoặc kép. Các thuộc tính có thể được sử dụng để phân biệt các phần tử trùng tên. Trong ví dụ ở Chương trình 2.1, ta thấy tên nhà hàng có tiếng Anh và tiếng Việt, lúc này nếu sử dụng thuộc tính, có thể gom hai thẻ về tên nhà hàng về một thẻ và có các thuộc tính khác nhau như sau:

```
<nhang>
    <ten ngonngu="TiengViet">
        Nhà hàng Hoàng Tâm
    </ten>
    <ten ngonngu="TiengAnh">
        Hoang Tam Restaurant
    </ten>
    ...

```

Như vậy, với cùng một thẻ `<ten>` nhưng có các thuộc tính khác nhau sẽ giúp phân biệt được các giá trị khác nhau.

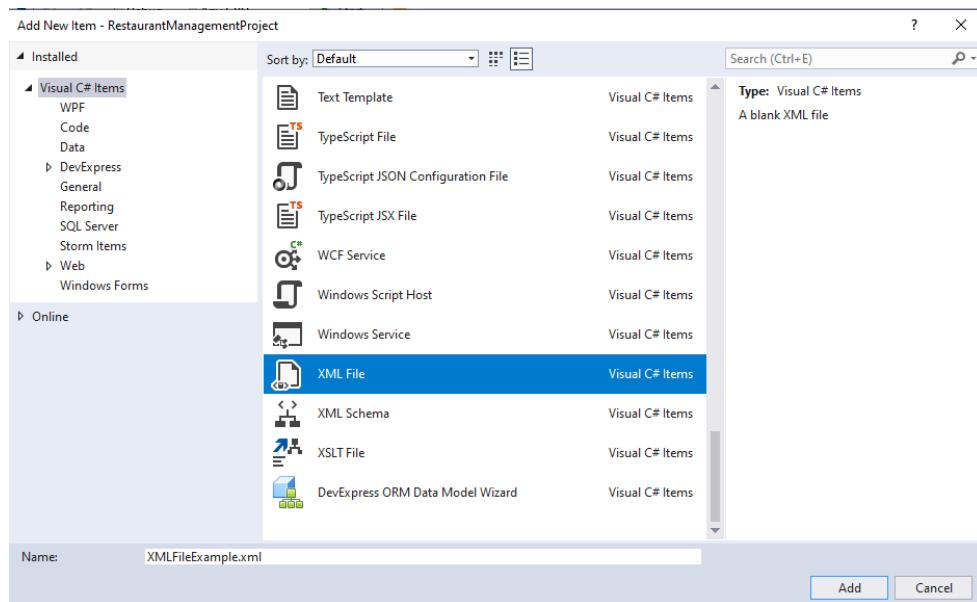
2.2.3. Cách tạo tập tin XML bằng Visual Studio

Visual Studio cho phép tạo tập tin XML để tích hợp vào dự án. Sau khi đã tạo dự án, muốn thêm một tập tin XML, click phải lên dự án, chọn Add, chọn New Item..., tìm đến tập tin XML (Hình 2.2), đặt tên tập tin và nhấn Add.

Tập tin XML luôn được bắt đầu bằng chỉ thị:

```
<?xml version="1.0" encoding="utf-8" ?>
```

Đây chính là chỉ thị cho phép khai báo các thông tin về tập tin như phiên bản hiện tại, chế độ mã hoá là UTF-8. Từ đây, bắt đầu khai báo các nút của tập tin XML, nút đầu tiên được gọi là nút gốc, sau đó tới các nút con như trong Hình 2.3.



Hình 2.2. Tạo tập tin XML

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <nhahang>
3      <ten_ngonngu="TiengViet">
4          Nhà hàng Hoàng Tâm
5      </ten>
6      <ten_ngonngu="TiengAnh">
7          Hoang Tam Restaurant
8      </ten>
9      <diachi>
10         221 Bà Triệu, Phường 4, TP. Đà Lạt
11     </diachi>
12     <dienthoai>
13         02633 555 247
14     </dienthoai>
15     <website>
16         www.hoangtamdl.vn
17     </website>
18 </nhahang>
```

Hình 2.3. Tập tin XML trong Visual Studio

Trong ngôn ngữ C#, để đọc được nội dung các thẻ trong tập tin XML có thể sử dụng các lớp được cung cấp sẵn bởi .NET Framework trong không gian tên System.Xml, thứ hai là sử dụng thư viện LINQ to XML với không gian tên System.Xml.Linq. Chi tiết về cách đọc XML có thể xem thêm trong Chương 5.

2.2.4. Tạo chuỗi kết nối trong tập tin App.config

Trong dự án vừa tạo, ta có thể thấy hệ thống tự thêm vào một tập tin có tên App.config, đây được gọi là tập tin cấu hình, chứa toàn bộ cấu hình của dự án. Các cấu hình của dự án bao gồm: Các tham số, các tính năng bảo mật và đặc biệt là chuỗi kết nối đến cơ sở dữ liệu. Cấu trúc của tập tin App.config chính là các thẻ dạng XML. Người dùng có thể thêm các tham số cấu hình trong tập tin này. Ở đây sẽ hướng dẫn thêm một chuỗi kết nối tới cơ sở dữ liệu.

Nhấp đôi chuột vào tập tin App.config trên dự án, thêm thẻ <connectionStrings> bên trong thẻ <configuration> như sau:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
    </startup>
    <connectionStrings>
        <add name="ConnectionStringName" connectionString="server=localhost;
            database=RestaurantManagement; integrated
            security=true;" />
    </connectionStrings>
</configuration>
```

Trong đó:

- *ConnectionStringName*: Là tên chuỗi kết nối, tên này do chúng ta tự đặt;
- *Localhost*: Là tên máy tính chứa cơ sở dữ liệu (mở SQL Server lên để xem tên của Server);
- *RestaurantManagement*: Là tên cơ sở dữ liệu cần kết nối tới;
- *integrated security=true*: Là dòng cho phép kết nối không sử dụng đến tài khoản và mật khẩu.

Để đọc được chuỗi kết nối này bằng ngôn ngữ C#, phải sử dụng thư viện System.Configuration, chi tiết được mô tả trong Chương 7 của giáo trình này.

2.3. Kiểu dữ liệu JSON

Một kiểu dữ liệu đơn giản khác cũng thường xuyên được sử dụng để lưu trữ dữ liệu, đó là JSON. JSON (*Javascript Object Notation*) là một dạng dữ liệu tuân theo một quy luật nhất định mà hầu hết các ngôn ngữ lập trình có thể đọc được. Người dùng có thể dùng kiểu dữ liệu này để lưu một mẫu tin, truy vấn và xuất ra giao diện kết quả. Hiện nay, JSON vì có cấu trúc đơn giản, dễ sử dụng nên được ứng dụng khá phổ biến, đặc biệt cho các ứng dụng trên Web. JSON cũng được dùng để tạo ra các tập tin API, là một dạng thức được quy định bởi các hàm mở, người dùng gọi các hàm đó để trả về kết quả và xử lý.

2.3.1. Cú pháp của JSON

Có thể dùng bất kỳ công cụ soạn thảo văn bản nào để soạn thảo tập tin JSON, tập tin dạng JSON có thể được lưu với nhiều kiểu mở rộng, tuy nhiên thông thường thì nó được lưu dưới dạng *.json hoặc *.js. Cú pháp của JSON tuân theo dạng cặp key-value, có hỗ trợ cấu trúc dữ liệu như đối tượng hoặc mảng, bao gồm các thành phần sau:

- Dữ liệu được biểu diễn dưới dạng cặp: key-value;
- Các dấu ngoặc nhọn lưu trữ tên các đối tượng, tiếp theo là dấu hai chấm, sau đó là giá trị; Các cặp tên/giá trị được phân tách bằng dấu phẩy; Đối tượng hoặc giá trị được để trong dấu ngoặc kép;
- Dấu ngoặc vuông biểu diễn kiểu dữ liệu mảng, các giá trị cũng được phân cách bằng dấu phẩy.

Ví dụ 1: Để lưu trữ các thông tin về nhà hàng như: Tên, địa chỉ, số điện thoại, website bằng tập tin JSON, ta tạo cấu trúc file JSON như sau:

```
{  
    "nhahang": {  
        "tentiengViet": "Nhà hàng Hoàng Tâm",  
        "tentiengAnh": "Hoang Tam restaurant",  
        "diachi": "221 Bà Triệu, Phường 4, TP. Đà Lạt",  
        "dienthoai": "02633 555 247",  
        "website": "www.hoangtamdl.vn"  
    }  
}
```

Theo cấu trúc trên, đầu tiên ta có một đối tượng “nhahang”, đối tượng này có một thực thể bao gồm các thuộc tính như: Tên tiếng Việt, tên tiếng Anh, địa chỉ, điện thoại, website và các giá trị đi kèm. Các cặp khóa, giá trị này được lưu trữ dưới dạng tập tin văn bản nên dễ đọc, dễ lưu trữ và xử lý.

Ví dụ 2: Nếu cần lưu trữ nhiều nhà hàng, ta có thể sử dụng cấu trúc mảng để lưu trữ, cấu trúc mảng được biểu diễn bởi cặp ngoặc vuông. Có thể lưu trữ thông tin về 2 nhà hàng như trong cấu trúc như sau:

```
{  
    "nhahang": [  
        {  
            "tentiengViet": "Nhà hàng Hoàng Tâm",  
            "tentiengAnh": "Hoang Tam restaurant",  
            "diachi": "221 Bà Triệu, Phường 4, TP. Đà Lạt",  
            "dienthoai": "02633 555 247",  
            "website": "www.hoangtamdl.vn"  
        },  
        {  
            "tentiengViet": "Nhà hàng Hoàng Tâm 2",  
            "tentiengAnh": "Hoang Tam 2 restaurant",  
        }  
    ]  
}
```

```

        "diachi": "250 Phan Đình Phùng, Phường 2, TP. Đà Lạt",
        "dienthoai": "02633 666 257",
        "website": "www.hoangtamdl.vn"
    }
]
}

```

Cũng theo cấu trúc trên, đối tượng “nhahang” sẽ bao gồm hai thực thể, mỗi một thực thể sẽ có các thuộc tính tương ứng. Cấu trúc dữ liệu mảng sẽ giúp lưu trữ được nhiều dữ liệu hơn.

2.3.2. Các kiểu dữ liệu trong JSON

JSON sử dụng nhiều kiểu dữ liệu để biểu diễn, kiểu dữ liệu được thể hiện trong phần giá trị của đối tượng. Các kiểu dữ liệu được thể hiện như trong Bảng 2.2.

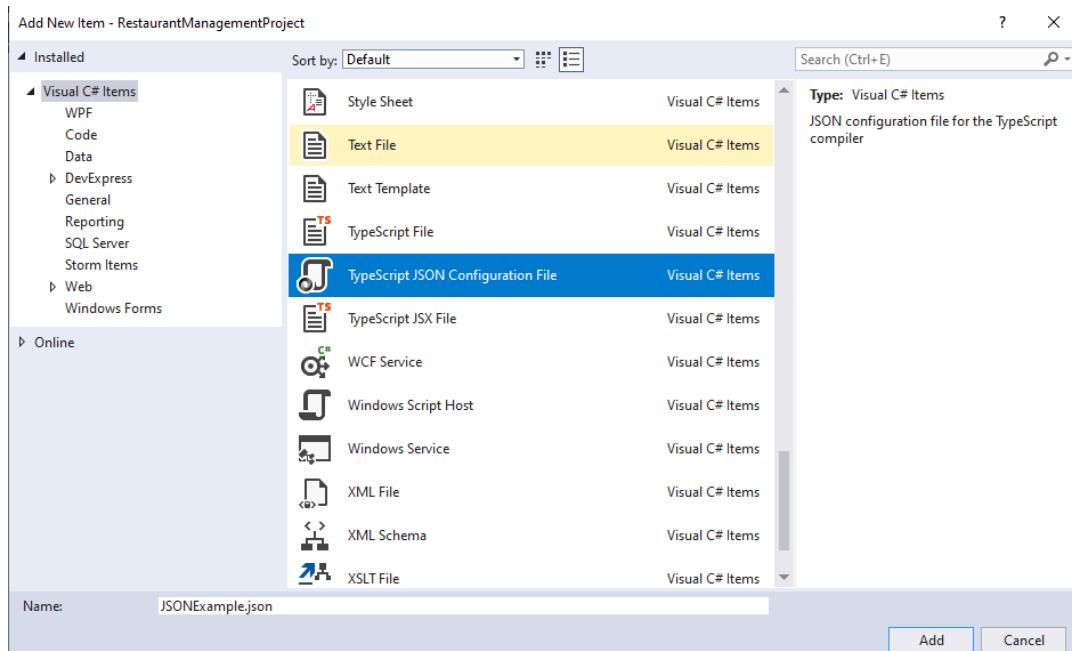
Bảng 2.2. Các kiểu dữ liệu trong JSON

Kiểu dữ liệu	Mô tả	Ví dụ
Number	Cho phép lưu trữ số nguyên, số thực, hoặc số mũ.	“sinhvien”: { “tuoi”: 20, “diem”: 7.5 }
String	Cho phép lưu trữ chuỗi ký tự, chuỗi ký tự phải nằm trong dấu nháy kép.	“sinhvien”: { “ho”: “Nguyễn”, “tenlot”: “Văn”, “ten”: “An” }
Boolean	Kiểu dữ liệu true hoặc false	“sinhvien”: { “tongiao”: false }
Array	Kiểu dữ liệu mảng, sử dụng dấu ngoặc vuông để biểu diễn nhiều dữ liệu	“sinhvien”: [{ “hoten”: “Nguyễn An”, “tuoi”: 19, “tongiao”: false }, { “hoten”: “Phạm Quốc”, “tuoi”: 20, “tongiao”: true }]
Object	Là một tập hợp các cặp key-value không theo thứ tự, các cặp này được đặt trong dấu ngoặc, key luôn là chuỗi và sau là dấu hai chấm	{ “id”: 1245213, “hoten”: “Nguyễn An”, “tuoi”: 20, “diemtongket”: 7.78 }

Ngoài các kiểu được nêu trên, JSON còn có thể sử dụng thêm các kiểu khác như khoảng trắng dùng để thêm giữa các key-value, hoặc giá trị null dùng để biểu diễn vô giá trị.

2.3.3. Cách tạo tập tin JSON bằng Visual Studio

Hiện tại, Visual Studio đã cho phép tạo các tập tin dạng JSON để tích hợp vào dự án. Sau khi đã tạo dự án Windows Form, muốn thêm một tập tin dạng JSON, click phải lên dự án, chọn Add, chọn New Item..., tìm đến mục TypeScript JSON Configuration File (Hình 2.4), đặt tên tập tin và nhấn Add.



Hình 2.4. Thêm tập tin JSON vào dự án

Khi đó, một tập tin dạng JSON được thêm vào và mở sẵn với một ví dụ mẫu, chúng ta xoá đi và viết lại hoặc thay đổi ví dụ mẫu thành tập tin lưu trữ thông tin là được (Hình 2.5).

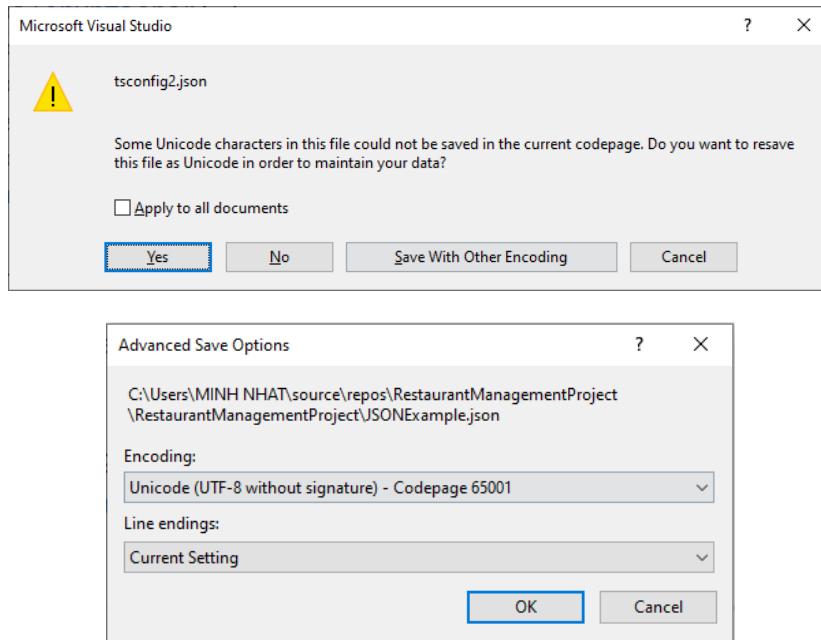
```
1 {  
2   "sinhvien": [  
3     {  
4       "MSSV": "1245732", "hoten": "Thái Duy Quý",  
5       "tuoi": 21, "diem": 8.5, "tongiao": false  
6     },  
7     {  
8       "MSSV": "1245872", "hoten": "Phan Thị Thanh Nga",  
9       "tuoi": 20, "diem": 9.0, "tongiao": true  
10    }  
11  ]  
12}
```

A screenshot of the 'JSONExample.json' file in Visual Studio. The code editor shows a JSON object with a single key 'sinhvien' containing an array of two student objects. Each student object has properties: MSSV, hoten, tuoi, diem, and tongiao. The code is color-coded for readability.

Hình 2.5. Tập tin JSON lưu trữ thông tin sinh viên

Một lưu ý khi tạo tập tin JSON là nếu trong đó gõ nội dung tiếng Việt (Unicode) thì khi lưu phải lưu dưới dạng Unicode, sau khi nhấn nút Save, hệ thống sẽ hỏi bởi một

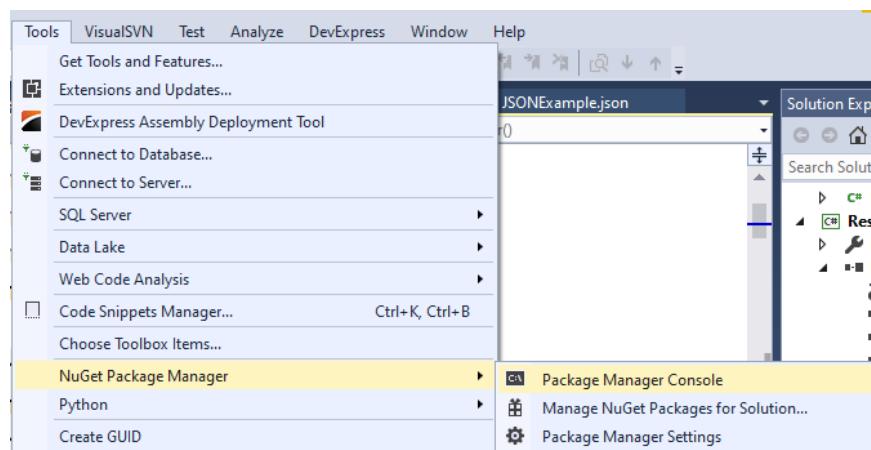
hộp thoại, chọn Save with Other Unicode và chọn trong mục Encoding là Unicode (UTF-8 without signature) (Hình 2.6).



Hình 2.6. Lưu tập tin JSON với Unicode UTF8

2.3.4. Đọc tập tin JSON bằng Visual Studio với ngôn ngữ C#

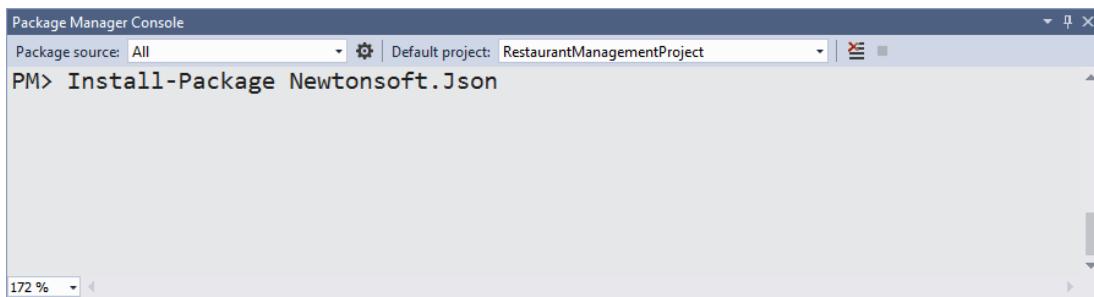
Để đọc được cấu trúc trong tập tin JSON bằng ngôn ngữ C#, chúng ta phải cài thêm gói thư viện Newtonsoft.Json vào dự án. Để cài gói thư viện này, thực hiện như sau: Vào menu Tools, chọn NuGet Package Manager, chọn Package Manager Console (Hình 2.7);



Hình 2.7. Cài đặt gói thư viện mở vào dự án

Khi đó cửa sổ Package Manager Console hiện ra phía dưới màn hình cho phép người dùng gõ lệnh (Hình 2.8); Từ đây có thể cài nhiều gói khác nhau vào dự án, để cài gói Newtonsoft.Json, chúng ta gõ lệnh sau và nhấn Enter:

```
Install-Package Newtonsoft.Json
```



Hình 2.8. Cửa sổ Package Manager Console

Hệ thống sẽ tìm phiên bản phù hợp nhất và cài vào dự án. Để sử dụng thư viện này và đọc được tập tin JSON, chúng ta sử dụng các câu lệnh using như sau:

```
using Newtonsoft.Json;
using System.IO;
using Newtonsoft.Json.Linq;
```

Giả sử cần đọc tập tin JSON như trong Hình 2.5, với cấu trúc sinh viên bao gồm các trường như: MSSV, họ tên, tuổi, điểm, và tôn giáo. Khi đó, để thuận tiện cần phải xây dựng một mô hình lớp bao gồm các trường như trong tập tin JSON, mô hình lớp có thể được xây dựng như trong lớp sau đây:

```
public class StudentInfo
{
    // Các thuộc tính
    public string MSSV { get; set; }
    public string Hoten { get; set; }
    public int Tuoi { get; set; }
    public double Diem { get; set; }
    public bool TonGiao { get; set; }
    // Phương thức tạo lập
    public StudentInfo(string mssv, string hoten, int tuoi, double diem,
bool tongiao)
    {
        this.MSSV = mssv;
        this.Hoten = hoten;
        this.Tuoi = tuoi;
        this.Diem = diem;
        this.TonGiao = tongiao;
    }
}
```

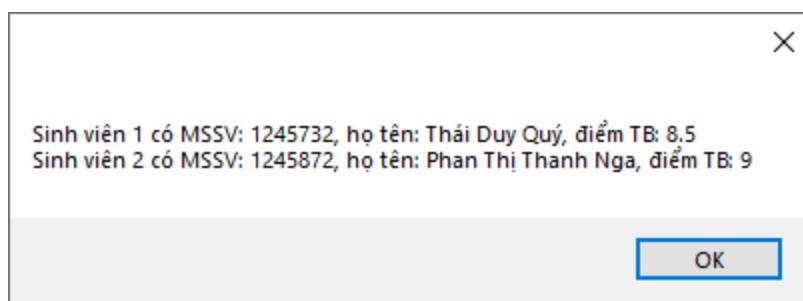
Để đọc được các nội dung trong tập tin JSON (ví dụ trong Hình 2.5), ta xây dựng phương thức đọc JSON như sau:

```
/// <summary>
/// Phương thức đọc tập tin JSON
/// </summary>
/// <param name="Path">Đường dẫn tập tin</param>
/// <returns>Danh sách các đối tượng từ tập tin JSON</returns>
private List<StudentInfo> LoadJSON(string Path)
```

```
// Khai báo danh sách lưu trữ
List<StudentInfo> List = new List<StudentInfo>();
// Đổi tượng đọc tập tin
StreamReader r = new StreamReader(Path);
string json = r.ReadToEnd(); // Đọc hết
// Chuyển về thành mảng các đối tượng
var array = (JObject)JsonConvert.DeserializeObject(json);
// Lấy đối tượng sinh viên
var students = array["sinhvien"].Children();
foreach (var item in students) // Duyệt mảng
{
    // Lấy các thành phần
    string mssv = item["MSSV"].Value<string>();
    string hoten = item["hoten"].Value<string>();
    int tuoi = item["tuoi"].Value<int>();
    double diem = item["diem"].Value<double>();
    bool tongiao = item["tongiao"].Value<bool>();
    // Chuyển vào đối tượng StudentInfo
    StudentInfo info = new StudentInfo(mssv, hoten, tuoi, diem,
    tongiao);
    List.Add(info); // Thêm vào danh sách
}
return List;
}
```

Phương thức ở trên có thể được viết trong lớp Form. Để đọc được tập tin JSON, chúng ta có thể tạo một nút và xử lý sự kiện Click như sau, kết quả sau khi nhấn nút được thể hiện như trên Hình 2.9.

```
private void btnJSON_Click(object sender, EventArgs e)
{
    string Str = ""; // chuỗi lưu trữ
    string Path = "../../JSONExample.json"; // Đường dẫn tập tin
    List<StudentInfo> List = LoadJSON(Path); // Gọi phương thức
    for (int i = 0; i < List.Count; i++) // Đọc danh sách
    {
        StudentInfo info = List[i];
        Str += string.Format("Sinh viên {0} có MSSV: {1}, họ tên: {2}, điểm TB: {3}\r\n", (i + 1), info.MSSV, info.Hoten, info.Diem);
    }
    MessageBox.Show(Str);
}
```



Hình 2.9. Kết quả khi đọc tập tin JSON

2.4. Kiểu dữ liệu BSON

BSON viết tắt của từ Binary JSON, là một kiểu mã hoá và lưu trữ dữ liệu dạng bán cấu trúc giống như JSON. Tuy vậy, kiểu dữ liệu BSON có hỗ trợ thêm kiểu nhị phân (*BinData*) và kiểu ngày tháng (*DateData*) để đa dạng hoá việc lưu trữ dữ liệu. Ngoài JSON, thì việc lưu trữ dữ liệu BSON có các đặc trưng như giúp lưu trữ nhẹ hơn, dễ duyệt hơn, và hiệu quả hơn trong quá trình mã hoá và giải mã. Kiểu dữ liệu này thường được dùng trong MongoDB, một hệ cơ sở dữ liệu nhẹ và hiệu quả hiện nay. Hiện tại các ngôn ngữ cơ bản như C, C#, Java, Python... đều đã hỗ trợ để xử lý loại dữ liệu này. Bảng 2.3 biểu diễn mối quan hệ giữa hai loại dữ liệu là JSON và BSON và Bảng 2.4 minh họa một số kiểu dữ liệu được sử dụng trong BSON.

Bảng 2.3. Mối quan hệ giữa JSON và BSON

JSON	BSON	Giải thích
{ "hello": "world" }	\x16\x00\x00\x00 // Kích thước tập tin hello\x00 // 0x02 là kiểu String \x06\x00\x00\x00world\x00 // trường khoá \x00 // trường giá trị // 0x00 là kiểu EOO ('kết thúc')	
{	\x31\x00\x00\x00 // Kích thước tập tin	
"BSON":	\x04BSON\x00 // Trường khoá	
[\x26\x00\x00\x00	
"awesome",	\x02\x30\x00\x08\x00\x00\x00awesom	// Các giá trị trong mảng
5.05,	e\x00	
1986	\x01\x31\x00\x33\x33\x33\x33\x33\x3	
]	3\x14\x40	
}	\x10\x32\x00\xc2\x07\x00\x00	
	\x00 //Kết thúc	
	\x00	

Bảng 2.4. Kiểu dữ liệu sử dụng trong BSON

Kiểu dữ liệu	Mô tả
unicode	Kiểu dữ liệu chuỗi, sử dụng UTF-8
integer	Kiểu số nguyên 32bit hoặc 64bit.
double	Kiểu dữ liệu thực, sử dụng dấu phẩy động 64bit của IEEE 754.
decimal	Kiểu số thập phân, theo chuẩn 128-bit IEEE 754-2008.
byte array	Kiểu dữ liệu mảng, sử dụng các byte để lưu trữ.
boolean	Kiểu bit, trả về true hoặc false
null	Kiểu null
object	Kiểu đối tượng, một đối tượng là một cặp khoá-giá trị
MD5 binary data	Kiểu mã hoá nhị phân MD5

Để sử dụng ngôn ngữ C# đọc dữ liệu BSON, chúng ta cũng phải cài đặt gói thư viện Newtonsoft.Json như trong JSON. Sau khi cài đặt, cần sử dụng không gian tên như sau:

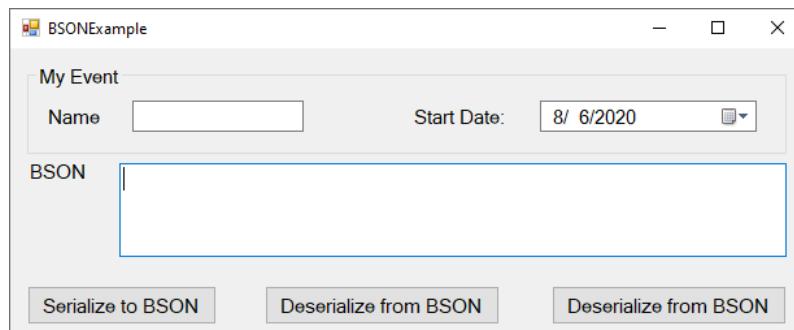
```
using Newtonsoft.Json.Bson;
```

Chương trình 2.2 minh họa cách sử dụng BSON với ba phương thức cơ bản: Chuyển đổi dữ liệu thường thành BSON (*SerializeToBSON*); Chuyển đổi dữ liệu từ BSON (*DeserializeFromBSON*); và phương thức chuyển đổi dữ liệu từ mảng BSON (*DeserializeBSONCollection*). Chương trình có xây dựng hai lớp, lớp MyEvent là mô hình biểu diễn dữ liệu với hai trường là Name và StartDate, lớp còn lại chứa các phương thức nêu trên. Ngoài ra còn có lớp Form để gọi các phương thức đó và xuất ra cửa sổ dạng MessageBox.

Chương trình 2.2. Minh họa cách sử dụng BSON bằng C#

```
// Lớp MyEvent với 2 trường là Name và StartDate
public class MyEvent
{
    public string Name { get; set; }
    public DateTime StartDate { get; set; }
}
// Lớp MyBSON chứa các phương thức xử lý
public class MyBSON
{
    /// <summary>
    /// Phương thức chuyển dữ liệu thành BSON
    /// </summary>
    /// <param name="myEvent">Đối tượng MyEvent</param>
    public string SerializeToBSON(MyEvent myEvent)
    {
        // Khai báo đối tượng MemoryStream và BsonWriter
        MemoryStream ms = new MemoryStream();
        BsonWriter writer = new BsonWriter(ms);
        // Khai báo đối tượng JsonSerializer để lấy dữ liệu
        JsonSerializer serializer = new JsonSerializer();
        serializer.Serialize(writer, myEvent); // Lấy dữ liệu
        // Xuất kết quả
        string data = Convert.ToBase64String(ms.ToArray());
        return data;
    }
    /// <summary>
    /// Phương thức lấy dữ liệu từ tập tin BSON
    /// </summary>
    /// <param name="JSONStr">Đoạn mã BSON</param>
    public MyEvent DeserializeFromBSON(string JSONStr)
    {
        // Mã hóa thành Byte Array
        byte[] data = Convert.FromBase64String(JSONStr);
        // Khai báo các đối tượng
        MemoryStream ms = new MemoryStream(data);
        BsonReader reader = new BsonReader(ms);
```

```
JsonSerializer serializer = new JsonSerializer();
// Thực hiện chuyển đổi dữ liệu
MyEvent myEvent = serializer.Deserialize<MyEvent>(reader);
return myEvent; // Trả về kết quả
}
/// <summary>
/// Phương thức lấy danh sách MyEvent từ tập tin BSON
/// </summary>
/// <param name="JSONStr">Đoạn mã BSON chứa tập dữ liệu</param>
public List<MyEvent> DeserializeBSONCollection(string JSONStr)
{
    // Mã hóa thành Byte Array
    byte[] data = Convert.FromBase64String(JSONStr);
    // Khai báo các đối tượng
    MemoryStream ms = new MemoryStream(data);
    BsonReader reader = new BsonReader(ms);
    JsonSerializer json = new JsonSerializer();
    // Gán thông số cho biết trả về dạng chuỗi
    reader.ReadRootValueAsArray = true;
    // Thực hiện việc chuyển đổi
    List<MyEvent> eventList = json.Deserialize<List<MyEvent>>(reader);
    return eventList; // Trả về kết quả
}
}
```



Hình 2.11. Chương trình Windows Form minh họa BSON

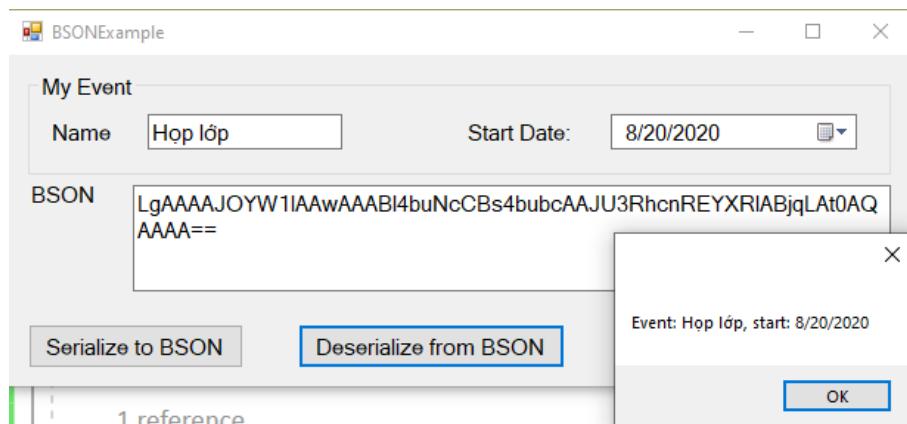
Để chạy các phương thức trên, chúng ta xây dựng chương trình Windows Form như Hình 2.11, sự kiện khi nhấn nút Serialize To BSON như sau:

```
private void btnSerializeToBSON_Click(object sender, EventArgs e)
{
    MyBSON bSON = new MyBSON();
    string name = txtName.Text;
    DateTime startDate = dateTime.Value;
    MyEvent myEvent = new MyEvent()
    {
        Name = name,
        StartDate = startDate
    };
    string result = bSON.SerializeToBSON(myEvent);
    txtBSON.Text = result;
}
```

Tương tự, sự kiện nút *Deserialize from BSON* được viết như sau:

```
private void btnDeserializeFromBSON_Click(object sender, EventArgs e)
{
    string BSONText = txtBSON.Text;
    MyBSON bSON = new MyBSON();
    MyEvent myEvent = bSON.DeserializeFromBSON(BSONText);
    string result = string.Format("Event: {0}, start: {1}",
        myEvent.Name, myEvent.StartDate.ToShortDateString());
    MessageBox.Show(result);
}
```

Sau khi nhấn nút *Serialize*, hệ thống sẽ lấy dữ liệu từ *TextBox* và *DatePicker* để mã hoá thành tập tin BSON và hiển thị trong ô *Text* ở phía dưới. Ngược lại, khi nhấn nút *Deserialize from BSON*, hệ thống sẽ dịch ngược và cho ra kết quả bởi *MessageBox* (Hình 2.12). Chương trình còn một nút bấm nữa, đọc giả tự xây dựng như là một bài tập.



Hình 2.12. Chạy chương trình minh họa BSON

2.5. Kết chương

Các hệ thống dữ liệu được phân chia thành ba loại là cấu trúc, phi cấu trúc và bán cấu trúc. Trong đó dữ liệu bán cấu trúc là dữ liệu đơn giản nhưng lại có những cấu trúc riêng biệt giúp lưu trữ và truy xuất một cách dễ dàng. Chương này cung cấp thêm một số kiểu dữ liệu dạng bán cấu trúc để ngoài việc lưu trữ bằng SQL Server, đọc giả có thể chọn một số cách lưu trữ khác nếu dữ liệu nhỏ.

Bài tập:

1. Tìm hiểu về phương pháp đọc và ghi tập tin XML bằng ngôn ngữ C#, xây dựng chương trình trên Windows Form để đọc một tập tin XML xuất ra dữ liệu dạng Nút.
2. Thiết kế tập tin XML và JSON để lưu trữ thông tin của 5 sinh viên bao gồm: Họ tên, MSSV, lớp, số điện thoại, email.
3. Xây dựng chương trình Windows Form để đọc nội dung của tập tin XML và JSON của bài tập 2.

4. Xây dựng chương trình lưu trữ cấu trúc mảng của tập tin BSON.
5. Xây dựng chương trình đọc tập tin BSON chứa mảng dữ liệu và đưa lên ListView.

CHƯƠNG 3. KẾT NỐI CƠ SỞ DỮ LIỆU

Chương này trình bày chi tiết cách sử dụng lớp Connection để kết nối đến một cơ sở dữ liệu. Có 3 lớp Connection thường được dùng: SqlConnection, OleDbConnection và OdbcConnection. Đối tượng SqlConnection được sử dụng để tạo một kết nối tới cơ sở dữ liệu SQL Server. Đối tượng OleDbConnection dùng để kết nối tới cơ sở dữ liệu có hỗ trợ chuẩn OLE DB như Oracle hay Access. Cuối cùng, đối tượng OdbcConnection dùng để kết nối tới cơ sở dữ liệu có hỗ trợ chuẩn ODBC. Như vậy, muốn tương tác với cơ sở dữ liệu, trước hết, phải tạo một kết nối thông qua đối tượng Connection.

Những vấn đề sẽ được đề cập trong chương này:

- Tạo đối tượng kết nối;
- Mở và đóng kết nối;
- Tô hợp kết nối – Connection Pool;
- Quản lý trạng thái kết nối;
- Quản lý các sự kiện trong quá trình kết nối.

3.1. Tạo đối tượng kết nối

3.1.1. Kết nối tới cơ sở dữ liệu SQL Server

Để kết nối tới một cơ sở dữ liệu SQL Server, ta dùng một đối tượng của lớp SqlConnection. Đối tượng này quản lý và điều khiển việc liên lạc giữa cơ sở dữ liệu và chương trình ứng dụng. Lớp SqlConnection chỉ được dùng cho SQL Server nhưng hầu hết các thuộc tính, phương thức và sự kiện trong lớp này (liệt kê trong Bảng 3.1, 3.2 và 3.3) cũng giống như trong các lớp OleDbConnection và OdbcConnection.

Bảng 3.1. Các thuộc tính của lớp SqlConnection

Thuộc tính	Kiểu	Mô tả
ConnectionString	string	Cho phép lấy hay gán chuỗi thông tin kết nối tới cơ sở dữ liệu.
ConnectionTimeout	int	Trả về thời gian chờ (tính bằng giây) trong khi chương trình cố gắng thiết lập một kết nối tới cơ sở dữ liệu. Giá trị mặc định là 15 giây.
Database	string	Trả về tên của cơ sở dữ liệu đã hoặc đang kết nối.
DataSource	string	Trả về tên của máy server đang chạy SQL Server chứa cơ sở dữ liệu.
State	ConnectionState	Trả về trạng thái hiện tại của kết nối: Broken, Closed, Connecting, Executing, Fetching hay Open.
WorkstationId	string	Trả về chuỗi chứa thông tin máy khách đang chạy ứng dụng kết nối tới SQL Server. Thuộc tính này cũng chỉ được áp dụng cho SqlConnection.

Bảng 3.2. Các phương thức của lớp SqlConnection

Phương thức	Kiểu trả về	Mô tả
BeginTransaction()	SqlTransaction	Hàm dùng để thông báo bắt đầu một giao dịch.
ChangeDatabase()	void	Thay đổi cơ sở dữ liệu muốn kết nối.
Close()	void	Đóng kết nối tới cơ sở dữ liệu.
CreateCommand()	SqlCommand	Tạo và trả về một đối tượng thực thi lệnh SqlCommand.
Open()	void	Mở một kết nối cơ sở dữ liệu với các thông tin thiết lập kết nối được chỉ ra trong thuộc tính ConnectionString.

Ta cũng có thể sử dụng các sự kiện để cho phép một đối tượng phát đi các thông điệp đến một đối tượng khác nhằm thông báo có điều gì đó đã xảy ra. Chẳng hạn, khi nhấp chuột lên một ứng dụng Windows, một sự kiện xảy ra hay còn gọi là phát sinh (fired).

Bảng 3.3. Các sự kiện của lớp SqlConnection

Sự kiện	Phát sinh sự kiện	Mô tả
StateChange	StateChangeEvent Handler	Phát sinh khi trạng thái kết nối bị thay đổi.
InfoMessage	SqlInfoMessageEventHandler	Phát sinh khi cơ sở dữ liệu trả về một cảnh báo hay một thông điệp nào đó.

Để kết nối tới một cơ sở dữ liệu SQL Server, ta dùng một đối tượng của lớp SqlConnection. Đối tượng này quản lý và điều khiển việc liên lạc giữa cơ sở dữ liệu và chương trình ứng dụng. Lớp SqlConnection chỉ được dùng cho SQL Server nhưng hầu hết các thuộc tính, phương thức và sự kiện trong lớp này cũng giống như trong các lớp OleDbConnection và OdbcConnection

Để tạo một đối tượng SqlConnection, ta dùng phương thức tạo lập SqlConnection() của nó. Phương thức này ở dạng quá tải hàm, nghĩa là có nhiều hàm cùng tên nhưng khác nhau về cách gọi hàm và truyền tham số. Các dạng quá tải của hàm này là:

```
SqlConnection ()  
SqlConnection (string connectionString)
```

Trong đó, connectionString là một chuỗi chứa thông tin chi tiết về kết nối cơ sở dữ liệu. Để tạo một đối tượng SqlConnection, sử dụng đoạn mã sau:

```
mySqlConnection.ConnectionString =  
    "server=localhost;database=RestaurantManagement;uid=sa;pwd=sa";  
SqlConnection mySqlConnection = SqlConnection (connectionString);
```

Hoặc đơn giản hơn là

```
SqlConnection mySqlConnection = SqlConnection  
("server=localhost;database= RestaurantManagement;uid=sa;pwd=sa");
```

Hoặc:

```
SqlConnection mySqlConnection = new SqlConnection();
```

Sau đó, gán chuỗi thông tin về kết nối cơ sở dữ liệu cho thuộc tính ConnectionString của đối tượng mySqlConnection. Ví dụ:

```
mySqlConnection.ConnectionString =
    "server=localhost;database= RestaurantManagement;uid=sa;pwd=sa";
```

Trong đó:

- *server*: Là tên của máy tính đang chạy SQL Server;
- *database*: Là tên cơ sở dữ liệu muốn kết nối;
- *uid*: Là tên người dùng cơ sở dữ liệu;
- *pwd*: Là mật khẩu tương ứng của người dùng.

Vì lý do bảo mật, không nên đặt tên người dùng và mật khẩu trực tiếp vào chuỗi kết nối. Thay vào đó, chương trình có thể yêu cầu người sử dụng thiết lập tên và mật khẩu trước khi chạy hoặc có thể sử dụng chế độ bảo mật tích hợp (integrated security). Một điều nữa cần nhớ là ta chỉ có thể gán giá trị cho thuộc tính ConnectionString khi kết nối đã được đóng.

```
sqlConnection.ConnectionString = "server=.;  
database=RestaurantManagement; Integrated Security=true;");
```

Ta có thể thiết lập thêm tùy chọn thời gian chờ kết nối (*connection timeout*) để chỉ ra số giây mà phương thức Open() chờ cho đến khi thiết lập được một kết nối tới cơ sở dữ liệu. Để làm được điều này, chỉ cần bổ sung thêm một giá trị *connection timeout* vào chuỗi kết nối như sau.

```
string connectionString ="server=.; database=RestaurantManagement;  
Integrated Security=true; connection timeout=10";
```

Thời gian chờ mặc định cho một kết nối là 15 giây. Nếu giá trị này được đặt là 0 thì kết nối sẽ có thời gian chờ là vô hạn. Vì thế, không nên đặt giá trị *connection timeout* là 0.

Trước khi bắt đầu đăng nhập vào Windows, chúng ta cần cung cấp tên đăng nhập và mật khẩu. Trong trường hợp dùng chế độ bảo mật tích hợp (*integrated security*), ta có thể bỏ qua tên đăng nhập và mật khẩu trong chuỗi kết nối và sử dụng luôn các thông tin đăng nhập trước đó để kết nối tới cơ sở dữ liệu SQL Server. Ta có thể sử dụng *integrated security* trong ứng dụng bằng cách thêm đoạn *intergrated security=SSPI* vào chuỗi kết nối. Ví dụ:

```
string connectionString = "server=.; database=RestaurantManagement;  
intergrated security=SSPI;";
```

Để ý thấy trong chuỗi kết nối này không có tên người dùng cơ sở dữ liệu và mật khẩu. Thay vào đó, tên và mật khẩu khi người dùng đăng nhập vào hệ điều hành sẽ được gửi sang SQL Server. Sau đó, SQL Server kiểm tra trong danh sách người dùng của nó xem thử bạn có được quyền truy xuất cơ sở dữ liệu hay không.

3.1.2. Kết nối cơ sở dữ liệu Access

Để kết nối tới một cơ sở dữ liệu Access, ta dùng đối tượng OleDbConnection - thay vì SqlConnection ở mục trước – với chuỗi kết nối có dạng sau:

```
provider= Microsoft.ACE.OLEDB.12.0; data source=databaseFile
```

trong đó:

- *data source*: là tên cơ sở dữ liệu cần kết nối.
- *provider*: là một chuẩn của trình xử lý giao dịch trong cơ sở dữ liệu. Với Access, giá trị này được đặt là Microsoft.Jet.OLEDB.4.0.
- *databaseFile*: là đường dẫn đến tập tin chứa cơ sở dữ liệu cần kết nối.

Ví dụ: chuỗi kết nối sau được dùng để kết nối tới cơ sở dữ liệu RestaurantManagement lưu trong tập tin RestaurantManagement.mdb, quản lý bởi Microsoft Office Access.

```
string connectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" +  
    "data source= D:\\Database\\RestaurantManagement.accdb;" +  
    "Persist Security Info=False;";
```

Để tạo một đối tượng OleDbConnection với chuỗi kết nối cho trước, sử dụng đoạn mã sau:

```
OleDbConnection myOleDbConnection =  
    new OleDbConnection(connectionString);
```

Chương trình 3.1 minh họa cách sử dụng một đối tượng OleDbConnection để kết nối tới cơ sở dữ liệu RestaurantManagement quản lý bởi MS Access và lấy một dòng từ bảng Food. Ngoài ra, ví dụ này còn sử dụng hai đối tượng khác là OleDbCommand và OleDbDataAdapter để thực thi truy vấn và đọc kết quả trả về từ cơ sở dữ liệu RestaurantManagement.

Chương trình 3.1: Sử dụng OleDbConnection để kết nối cơ sở dữ liệu Access

```
using System;  
using System.Data.OleDb;  
class OleDbConnectionAccess  
{  
    public static void Main()  
    {  
        // khai báo chuỗi kết nối  
        string connectionString =  
            "Provider=Microsoft.ACE.OLEDB.12.0;" +
```

```
"data source= D:\\Database\\RestaurantManagement.accdb;" +
    Persist Security Info=False;";
// tạo đối tượng SqlConnection kết nối tới cơ sở dữ liệu, truyền vào
// chuỗi kết nối
OleDbConnection myOleDbConnection =
    new OleDbConnection(connectionString);
// tạo một đối tượng OleDbCommand
OleDbCommand myOleDbCommand = myOleDbConnection.CreateCommand();

// thiết lập thuộc tính CommandText của đối tượng OleDbCommand
// là một truy vấn SQL lấy danh sách thức ăn trong bảng Food theo loại
myOleDbCommand.CommandText =
    "SELECT * FROM Food WHERE FoodCategoryID = 1";
// Mở kết nối tới CSDL
myOleDbConnection.Open();
// Tạo đối tượng OleDbDataReader và gọi phương thức ExecuteReader()
// của đối tượng OleDbCommand để chạy câu lệnh truy vấn
OleDbDataReader myOleDbDataReader = myOleDbCommand.ExecuteReader();
// đọc từng dòng của đối tượng OleDbDataReader sử dụng
// phương thức Read()
myOleDbDataReader.Read();
// Hiển thị nội dung một số cột thông tin kết quả
Console.WriteLine("Ten mon an "] = " +
    myOleDbDataReader["Name"]);
Console.WriteLine("Don vi tinh "] = " +
    myOleDbDataReader["Unit"]);
Console.WriteLine("Don gia"] = " +
    myOleDbDataReader["Price"]);
// Đóng đối tượng OleDbDataReader sử dụng phương thức Close()
myOleDbDataReader.Close();
// Đóng đối tượng OleDbConnection sử dụng phương thức Close()
myOleDbConnection.Close();
}
}
```

Kết quả khi chạy chương trình:

```
Ten mon an : Bia Heniken
Don vi tinh : Chai
Don gia : 20000
```

3.1.3. Kết nối cơ sở dữ liệu Oracle

Để kết nối tới một cơ sở dữ liệu Oracle, ta cũng dùng đối tượng OleDbConnection với chuỗi kết nối có dạng:

```
provider=MSDAORA;data source=OracleNetServiceName;user id=username;
password=password
```

Trong đó:

- *OracleNetServiceName*: Là tên của dịch vụ Oracle Net cho cơ sở dữ liệu. Oracle Net là một thành phần của phần mềm cho phép bạn kết nối tới một cơ sở dữ liệu qua mạng. Bạn cần phải liên hệ với người quản trị cơ sở dữ liệu để biết tên của dịch vụ Oracle Net này.
- *User ID*: Là tên của người dùng được phép truy cập tới cơ sở dữ liệu.
- *Password*: Là mật khẩu tương ứng của người dùng.

Ví dụ sau tạo một chuỗi kết nối với tên là connectionString theo định dạng trên. Sau đó, dùng lớp OleDbConnection để tạo một kết nối tới một cơ sở dữ liệu Oracle.

```
string connectionString = "provider=MSDAORA;data source=ORCL; user id= Scott;password= Tiger";  
  
OleDbConnection myOleDbConnection =  
    new OleDbConnection(connectionString);
```

Tên người dùng Scotch và mật khẩu Tiger là những giá trị mặc định khi truy xuất đến những cơ sở dữ liệu mẫu của Oracle.

3.1.4. Kết nối cơ sở dữ liệu MySQL

Để kết nối tới một cơ sở dữ liệu MySQL, ta cũng dùng đối tượng OleDbConnection với chuỗi kết nối có dạng:

```
string connectionString =  
    " Provider=MySQLProv;dataSource=mydb;  
        user id=myUsername;password=myPassword;";  
  
OleDbConnection myOleDbConnection =  
    new OleDbConnection(connectionString);
```

Có thể tham khảo thêm các chuỗi kết nối dùng các đối tượng khác tại trang web <https://www.connectionstrings.com/mysql/>

3.1.5. Kết nối cơ sở dữ liệu SQLite

Để kết nối tới một cơ sở dữ liệu SQLite, ta dùng đối tượng OdbcConnection với chuỗi kết nối có dạng:

```
string connectionString =  
    "DRIVER=SQLite3 ODBC driver; Database=c:\mydb.db;  
    LongNames=0;Timeout=1000;NoTXN=0;SyncPragma=NORMAL;StepAPI=0;";  
  
OdbcConnection myOleDbConnection =  
    new OdbcConnection (connectionString);
```

Có thể tham khảo thêm các chuỗi kết nối dùng các đối tượng khác tại trang web <https://www.connectionstrings.com/sqlite/>

3.1.6. Kết nối tới file *.xlsx

Để kết nối tới một tập tin XLSX, ta cũng dùng đối tượng OdbcConnection với chuỗi kết nối có dạng:

```
string connectionString = "Provider=Microsoft.ACE.OLEDB.12.0;
                           Data source=D:\\Database\\Food.xlsx;
                           Extended Properties=\"Excel 12.0 Xml;HDR=YES\";
```

Có thể tham khảo thêm các chuỗi kết nối dùng các đối tượng khác tại trang web <https://www.connectionstrings.com/excel/>

3.2. Mở và đóng kết nối

Một khi đã tạo được đối tượng *Connection* và đặt giá trị thích hợp cho thuộc tính *ConnectionString*, cần phải mở kết nối tới cơ sở dữ liệu trước khi thực thi một lệnh truy vấn nào đó. Điều này được thực hiện bằng cách gọi hàm *Open()* của đối tượng *Connection* như sau:

```
mySqlConnection.Open();
```

Sau khi mở kết nối và thực hiện các giao dịch với cơ sở dữ liệu, cần phải đóng kết nối và giải phóng các tài nguyên bằng cách gọi phương thức *Close()* của đối tượng *Connection*.

```
mySqlConnection.Close();
```

Chương trình 3.2 minh họa các kết nối tới cơ sở dữ liệu RestaurantManagement của SQL Server sử dụng đối tượng *SqlConnection* và hiển thị giá trị của một vài thuộc tính của nó.

Chương trình 3.2. Dùng SqlConnection để kết nối cơ sở dữ liệu SQL Server

```
using System;
using System.Data.SqlClient;
class MySqlConnection
{
    public static void Main()
    {
        // chi tiết chuỗi kết nối
        string connectionString = "server=.;"
                               + "database=RestaurantManagement; Integrated Security=true; ";
        // tạo đối tượng SqlConnection kết nối tới cơ sở dữ liệu, truyền vào
        // chuỗi kết nối
        SqlConnection mySqlConnection =
            new SqlConnection(connectionString);
        // mở kết nối tới cơ sở dữ liệu dùng phương thức Open()
        mySqlConnection.Open();
        // hiển thị các thuộc tính của đối tượng SqlConnection
        Console.WriteLine("mySqlConnection.ConnectionString = " +
            mySqlConnection.ConnectionString);
        // hiển thị thời gian chờ khi kết nối tới CSDL, mặc định là 15s
```

```
Console.WriteLine("mySqlConnection.ConnectionTimeout = " +
    mySqlConnection.ConnectionTimeout);
// hiển thị tên của cơ sở dữ liệu đang kết nối
Console.WriteLine("mySqlConnection.Database = " +
    mySqlConnection.Database);
// hiển thị tên của máy server đang chạy SQL Server chứa cơ sở dữ liệu.
Console.WriteLine("mySqlConnection.DataSource = " +
    mySqlConnection.DataSource);
// hiển thị kích thước (tính theo byte) của các gói tin trong mạng
dùng để liên lạc với SQL Server.
// Thuộc tính này chỉ áp dụng cho SqlConnection. Giá trị mặc định là
8192 bytes.
Console.WriteLine("mySqlConnection.PacketSize = " +
    mySqlConnection.PacketSize);
// hiển thị phiên bản của SQL Server
Console.WriteLine("mySqlConnection.ServerVersion = " +
    mySqlConnection.ServerVersion);
// hiển thị trạng thái hiện tại của kết nối: Broken, Closed, Connecting,
Executing, Fetching hay Open.
Console.WriteLine("mySqlConnection.State = " +
    mySqlConnection.State);
// hiển thị chuỗi chứa thông tin máy khách đang chạy ứng dụng kết nối
tới SQL Server. Thuộc tính này cũng chỉ được áp dụng cho SqlConnection
Console.WriteLine("mySqlConnection.WorkstationId = " +
    mySqlConnection.WorkstationId);
// đóng kết nối sử dụng phương thức Close();
mySqlConnection.Close();
}
}
```

Kết quả sau khi chạy chương trình:

```
mySqlConnection.ConnectionString = server=.; database =
RestaurantManagement; Integrated Security = true;
mySqlConnection.ConnectionTimeout = 15
mySqlConnection.Database = RestaurantManagement
mySqlConnection.DataSource = .
mySqlConnection.PacketSize = 8000
mySqlConnection.ServerVersion = 11.00.2218
mySqlConnection.State = Open
mySqlConnection.WorkstationId = THANHNGAPC
```

3.3. Tổ hợp kết nối – *Connection Pooling*

Việc mở và đóng một kết nối tới cơ sở dữ liệu là quá trình tương đối tốn thời gian. Vì thế, ADO .Net tự động lưu các kết nối cơ sở dữ liệu vào một tổ hợp (*pool*). Tổ hợp kết nối nhằm cải thiện hiệu suất một cách đáng kể vì không cần phải chờ khi tạo một kết nối mới tới cơ sở dữ liệu đã từng thiết lập kết nối trước đó. Khi đóng kết nối bởi phương thức Close, nó không thực sự đóng hẳn, thay vào đó, kết nối sẽ được đánh dấu là không được sử dụng và được lưu vào tổ hợp. Như vậy, nó luôn sẵn sàng cho lần kết nối sau.

Sau đó, nếu có một yêu cầu kết nối với cùng một thông tin (chẳng hạn như cùng tên cơ sở dữ liệu, tên đăng nhập và mật khẩu,...), hệ thống sẽ tìm trong tổ hợp một kết nối đang rảnh và trả về kết nối đó. Chúng ta dùng kết nối này để truy xuất đến cơ sở dữ liệu. Khi sử dụng đối tượng SqlConnection, có thể chỉ ra số kết nối tối đa, tối thiểu được phép lưu trong tổ hợp bằng cách thiết lập giá trị cho thuộc tính *max pool size*, *min pool size* trong chuỗi kết nối. Giá trị mặc định lần lượt là 100 và 0.

Ví dụ sau tạo một đối tượng kết nối SqlConnection với số kết nối tối đa trong tổ hợp là 10 và số kết nối tối thiểu là 5.

```
SqlConnection mySqlConnection =
    new SqlConnection( "server=.; database=RestaurantManagement;
                      Integrated Security=true;" +
                      "max pool size=10;min pool size=5");
```

Trong ví dụ này, một tổ hợp được tạo ra để chứa tối thiểu 5 và tối đa 10 đối tượng SqlConnection. Nếu cố gắng mở thêm một đối tượng SqlConnection mới trong khi tổ hợp đã đầy, yêu cầu mở sẽ phải chờ cho đến khi có một kết nối cũ bị đóng lại và lúc đó, kết nối vừa bị đóng sẽ được trả về yêu cầu mới. Nếu thời gian chờ lớn hơn số giây được quy định trong thuộc tính ConnectionTimeout, một lỗi sẽ phát sinh.

Chương trình 3.3 minh họa việc sử dụng tổ hợp kết nối nhằm tiết kiệm thời gian mở một kết nối mới dựa trên các kết nối đã đóng trong tổ hợp.

Chương trình 3.3. Connection Pooling

```
using System;
using System.Data.SqlClient;

class ConnectionPooling
{
    public static void Main()
    {
        // Tạo đối tượng SqlConnection kết nối tới CSDL
        // thiết lập tham số max pool size = 10 và min pool size = 5

        SqlConnection mySqlConnection =
            new SqlConnection("server=.;database=RestaurantManagement;" +
Integrated Security=true; max pool size=10;min pool size=5");
        // Mở đối tượng SqlConnection 10 lần

        for (int count = 1; count <= 10; count++)
        {
            Console.WriteLine("count = " + count);
            // Tạo đối tượng DateTime và thiết lập giá trị bằng thời gian
            // hiện tại
            DateTime start = DateTime.Now;
            // Mở kết nối tới cơ sở dữ liệu dùng phương thức Open()
            mySqlConnection.Open();
            // Trừ thời gian hiện tại với thời gian bắt đầu
            // Lưu khoảng thời gian cách biệt trong một đối tượng TimeSpan
            TimeSpan timeTaken = DateTime.Now - start;
```

```
// hiển thị khoảng thời gian (mili giây) cần để mở kết nối  
Console.WriteLine("Milliseconds = " + timeTaken.Milliseconds);  
// hiển thị trạng thái kết nối  
Console.WriteLine("mySqlConnection.State = " +  
    mySqlConnection.State);  
// Đóng kết nối  
mySqlConnection.Close();  
Console.ReadKey();  
}  
}  
}
```

Kết quả sau khi chạy chương trình

```
count = 1  
Milliseconds = 264  
mySqlConnection.State = Open  
count = 2  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 3  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 4  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 5  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 6  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 7  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 8  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 9  
Milliseconds = 0  
mySqlConnection.State = Open  
count = 10  
Milliseconds = 0  
mySqlConnection.State = Open
```

Với ví dụ này, ta có thể thấy thời gian để mở kết nối đầu tiên là 264 mili giây, khá dài so với các kết nối sau đó (0 mili giây). Sở dĩ có điều này là bởi vì kết nối đầu tiên sẽ phải tạo ra một kết nối thực sự tới cơ sở dữ liệu. Khi kết nối này bị đóng, nó được lưu vào tổ hợp kết nối. Khi có yêu cầu mở một kết nối mới, nó được lấy từ tổ hợp và việc này được thực hiện rất nhanh.

3.4. Quản lý trạng thái kết nối

Trạng thái của kết nối cho bạn biết quá trình yêu cầu kết nối tới cơ sở dữ liệu. Hai trạng thái dễ nhận biết nhất là Open và Closed. Để biết trạng thái hiện tại của một kết nối tới cơ sở dữ liệu, ta dùng thuộc tính *State* của đối tượng *SqlConnection*. Thuộc tính này trả về một hằng số có kiểu liệt kê (*enum*) *ConnectionState*. Một enum là một danh sách các hằng số, mỗi hằng số được đặt một tên. Bảng 3.4 liệt kê các hằng số được định nghĩa trong enum *ConnectionState*.

Bảng 3.4. Các trạng thái kết nối

Trạng thái	Mô tả
Broken	Kết nối đã bị ngắt vì một lý do nào đó. Điều này có thể xảy ra sau khi bạn mở kết nối. Để khắc phục, ta gọi hàm đóng kết nối và mở lại kết nối đó.
Closed	Kết nối đã bị đóng.
Connecting	Đang thiết lập kết nối tới cơ sở dữ liệu.
Executing	Kết nối đang được dùng trong quá trình thực thi một lệnh truy vấn.
Fetching	Kết nối đang được dùng trong quá trình nhận thông tin trả về từ cơ sở dữ liệu.
Open	Kết nối đang mở.

Với một ứng dụng tương đối lớn và phức tạp hoặc trong trường hợp muốn sử dụng cùng một kết nối ở nhiều nơi trong chương trình, cần phải kiểm tra trạng thái của kết nối trước khi mở. Điều này thực sự cần thiết vì nếu gọi phương thức *Open()* trên một kết nối đang mở sẽ phát sinh ra lỗi. Việc kiểm tra trạng thái kết nối có thể thực hiện như sau:

```
if (mySqlConnection.State == ConnectionState.Closed)
{
    mySqlConnection.Open();
}
```

3.5. Quản lý sự kiện trong quá trình kết nối

Các lớp Connection có hai sự kiện: *StateChange* và *InfoMessage*.

3.5.1. Sự kiện *StateChange*

Sự kiện này phát sinh khi trạng thái của kết nối bị thay đổi. Bạn cũng có thể sử dụng sự kiện này để quản lý các thay đổi về trạng thái của đối tượng *Connection*. Phương thức dùng để xử lý một sự kiện gọi là một trình xử lý sự kiện (*event handler*). Nó được gọi khi một sự kiện được phát sinh. Mọi phương thức để xử lý sự kiện đều phải có kiểu trả về là *void* và chấp nhận hai tham số đầu vào.

Tham số thứ nhất là đối tượng phát sinh ra sự kiện và có kiểu *object*. Lớp *System.Object* đóng vai trò là lớp cơ sở cho mọi lớp khác. Nói cách khác, mọi lớp khác đều được dẫn xuất từ lớp này. Tham số thứ hai là một đối tượng được tạo từ một lớp dẫn xuất từ lớp *System.EventArgs*. Lớp *EventArgs* là lớp cơ sở cho dữ liệu đi kèm trong

sự kiện và biểu diễn chi tiết về sự kiện đó. Trong trường hợp sự kiện StateChange, tham số thứ hai này là một đối tượng của lớp StateChangeEventArgs.

Ví dụ sau định nghĩa một phương thức có tên StateChangeHandler để xử lý sự kiện StateChange. Để biết trạng thái ban đầu và trạng thái hiện tại của kết nối, ta sử dụng thuộc tính OriginalState và CurrentState của tham số thứ hai.

```
public static void StateChangeHandler(
    object mySender, StateChangeEventArgs myEvent)
{
    Console.WriteLine(
        "mySqlConnection State has changed from " +
        myEvent.OriginalState + "to " +
        myEvent.CurrentState
    );
}
```

Để quản lý một sự kiện, chúng ta phải đăng ký trình xử lý cho sự kiện đó. Đoạn mã sau đăng ký phương thức StateChangeHandler() làm trình xử lý cho sự kiện StateChange của đối tượng SqlConnection.

```
mySqlConnection.StateChange +=
    new StateChangeEventHandler(StateChangeHandler);
```

Bất cứ khi nào sự kiện StateChange phát sinh, phương thức StateChangeHandler sẽ được gọi để cho biết trạng thái cũ và trạng thái hiện hành của kết nối. Chương trình 3.4 minh họa cách sử dụng sự kiện StateChange của lớp SqlConnection.

Chương trình 3.4. Sử dụng sự kiện StateChange

```
using System;
using System.Data.SqlClient;

class StateChange
{
    // Định nghĩa phương thức StateChangeHandler() để xử lý
    // các sự kiện StateChange
    public static void StateChangeHandler(
        object mySender, StateChangeEventArgs myEvent )
    {
        Console.WriteLine(
            "mySqlConnection State has changed from " +
            myEvent.OriginalState + " to " +
            myEvent.CurrentState
        );
    }
    public static void Main()
    {
        // Tạo đối tượng SqlConnection
        SqlConnection mySqlConnection = new
        SqlConnection("server=.;database=RestaurantManagement;" +
        "Integrated Security=true;");
```

```
// Quản lý các sự kiện StateChange sử dụng phương thức
//StateChangeHandler()
mySqlConnection.StateChange +=  
    new StateChangeEventhandler(StateChangeHandler);  
  
// Mở kết nối, trạng thái kết nối sẽ chuyển từ Closed sang Open
Console.WriteLine("Calling mySqlConnection.Open()");
mySqlConnection.Open();  
  
// Đóng kết nối, trạng thái kết nối sẽ chuyển từ Open sang Closed
Console.WriteLine("Calling mySqlConnection.Close()");
mySqlConnection.Close();  
  
Console.ReadKey();
}  
}
```

Kết quả sau khi chạy chương trình:

```
Calling mySqlConnection.Open()
mySqlConnection State has changed from Closed to Open
Calling mySqlConnection.Close()
mySqlConnection State has changed from Open to Closed
```

3.5.2. Sự kiện InfoMessage

Sự kiện này phát sinh khi cơ sở dữ liệu trả về một cảnh báo hoặc một thông điệp được sinh ra bởi cơ sở dữ liệu. Sự kiện InfoMessage được dùng để quản lý các thông điệp này. Để lấy thông điệp, ta đọc nội dung thuộc tính Errors của tham số SqlInfoMessageEventArgs. Chúng ta có thể tạo ra các thông báo lỗi hoặc thông tin nào đó bằng cách dùng mệnh đề RAISERROR hoặc PRINT của SQL Server.

Phương thức InfoMessageHandler() sau được dùng để xử lý sự kiện InfoMessage. Thuộc tính Errors của tham số thứ hai SqlInfoMessageEventArgs chứa các thông điệp phát sinh bởi cơ sở dữ liệu.

```
public static void InfoMessageHandler(
    object mySender, SqlInfoMessageEventArgs myEvent)
{
    Console.WriteLine(
        "The following message was produced:\n" +
        myEvent.Errors[0]
    );
}
```

Nếu muốn sử dụng trình cung cấp OLE DB, ODBC thay cho SQL, thay SqlInfoMessageEventArgs bởi OleDbInfoMessageEventArgs, OdbcInfoMessageEventArgs. Chương trình 3.5 minh họa cách sử dụng sự kiện InfoMessage. Chương trình này có sử

dụng phương thức ExecuteNonQuery() của đối tượng SqlCommand để gửi câu lệnh PRINT và RAISERROR tới cơ sở dữ liệu.

Chương trình 3.5. Sử dụng sự kiện InfoMessage

```
using System;
using System.Data;
using System.Data.SqlClient;

class InfoMessage
{
    // Định nghĩa phương thức InfoMessageHandler() để xử lý
    // các sự kiện InfoMessage
    public static void InfoMessageHandler(
        object mySender, SqlInfoMessageEventArgs myEvent )
    {
        Console.WriteLine(
            "The following message was produced:\n" +
            myEvent.Errors[0]
        );
    }
    public static void Main()
    {
        // Tạo một đối tượng SqlConnection object
        SqlConnection mySqlConnection = new
        SqlConnection("server=.;database=RestaurantManagement;" +
        "Integrated Security=true;");
        // Quản lý các sự kiện InfoMessage dùng phương thức
        // InfoMessageHandler()
        mySqlConnection.InfoMessage +=
            new SqlInfoMessageEventHandler(InfoMessageHandler);
        // Mở mySqlConnection
        mySqlConnection.Open();

        // Tạo một đối tượng SqlCommand
        SqlCommand mySqlCommand = mySqlConnection.CreateCommand();

        // Thực thi câu lệnh PRINT
        mySqlCommand.CommandText =
            "PRINT 'This is the message from the PRINT statement'";
        mySqlCommand.ExecuteNonQuery();

        // Chạy câu lệnh RAISERROR
        mySqlCommand.CommandText =
            "RAISERROR('This is the message from the RAISERROR
            statement', 10, 1)";
        mySqlCommand.ExecuteNonQuery();

        // Đóng kết nối
        mySqlConnection.Close();
        Console.ReadKey();
    }
}
```

Kết quả chạy chương trình:

```
The following message was produced:  
System.Data.SqlClient.SqlError: This is the message from the PRINT  
statement  
The following message was produced:  
System.Data.SqlClient.SqlError: This is the message from the  
RAISERROR statement
```

3.6. Kết chương

Để kết nối tới một hệ quản trị cơ sở dữ liệu, cần phải có một lớp trung gian để nhận chuỗi kết nối sau đó mở và đóng kết nối. Chương này đã trình bày chi tiết cách sử dụng các lớp Connection như SqlConnection, OleDbConnection và OdbcConnection để kết nối đến một cơ sở dữ liệu tương ứng. Tuỳ vào đặc trưng của ứng dụng, có thể chọn lớp tương ứng để kết nối. Tìm hiểu cách sử dụng của các lớp kết nối giúp người đọc dễ dàng kết nối tới các hệ thống, từ đó làm nền tảng để thực hiện truy vấn cho các chương tiếp theo.

Bài tập

1. Sử dụng lớp SqlConnection, thực hiện kết nối tới cơ sở dữ liệu SQL Server trên môi trường Windows Form bằng cách khai báo chuỗi kết nối kiểu String
2. Thực hiện như Câu 1 nhưng chuyển chuỗi kết nối vào tập tin App.Config
3. Sử dụng OleDbConnection và Windows Form, kết nối tới Cơ sở dữ liệu Access và Excel
4. Xây dựng ứng dụng cho phép người dùng kết nối vào SQL Server, chọn 1 Database và nhấn nút mở hoặc đóng kết nối
5. Xây dựng minh họa với Connection Pooling trên Form
6. Xây dựng minh họa trên Form với StateChange, InfoMessage.

CHƯƠNG 4. THỰC THI LỆNH THAO TÁC TRÊN CƠ SỞ DỮ LIỆU

Chương này giới thiệu và hướng dẫn thực thi các lệnh thao tác trên cơ sở dữ liệu bao gồm truy vấn cơ sở dữ liệu, lưu trữ dữ liệu, trích lọc và sắp xếp dữ liệu trả về, quản lý quan hệ và ràng buộc. Các lệnh truy vấn cơ sở dữ liệu được thực thi bởi đối tượng Command. Lớp Command cũng là một trong những lớp kết nối (*managed provider*). Lớp DataReader được dùng để đọc kết quả trả về từ cơ sở dữ liệu bởi đối tượng Command. Các đối tượng DataSet cho phép lưu trữ một bản sao thông tin của cơ sở dữ liệu. Từ đó, chúng ta có thể xử lý trực tiếp trên các thông tin này trong khi kết nối đã bị ngắt. Chương này còn giới thiệu về cách trích lọc và sắp xếp các dòng trong một DataTable dùng đối tượng DataView. Ưu điểm của DataView là có thể gắn (bind) nó cho một thành phần giao diện của Windows Form (chẳng hạn như điều khiển DataGridView).

Những nội dung chính sẽ được trình bày trong chương này bao gồm:

- Thao tác truy vấn cơ sở dữ liệu và đọc kết quả trả về;
- Lưu trữ dữ liệu với Dataset;
- Trích lọc và sắp xếp dữ liệu với DataView;

4.1. Thao tác truy vấn cơ sở dữ liệu và đọc kết quả trả về

Các lệnh truy vấn cơ sở dữ liệu được thực thi bởi đối tượng Command. Lớp Command cũng là một trong những lớp kết nối (*managed provider*). Có ba lớp Command: SqlCommand, OleDbCommand và OdbcCommand. Đối tượng Command được dùng để thực thi một lệnh truy vấn SQL như SELECT, INSERT, UPDATE hay DELETE. Nó cũng có thể được dùng để gọi một thủ tục (Stored Procedure) hay lấy tất cả các dòng, các cột từ một bảng cụ thể - lệnh này còn được gọi là TableDirect. Đối tượng Command liên lạc với cơ sở dữ liệu thông qua một đối tượng Connection.

Lớp DataReader được dùng để đọc kết quả trả về từ cơ sở dữ liệu bởi đối tượng Command. Việc đọc các dòng trong tập kết quả dùng đối tượng DataReader thường nhanh hơn đọc dữ liệu từ một DataSet. Các đối tượng DataReader đều thuộc lớp kết nối, có ba lớp DataReader: SqlDataReader, OleDbDataReader và OdbcDataReader. Các đối tượng DataReader chỉ đọc các dòng trong tập kết quả theo một chiều từ đầu đến cuối. Nó có thể dùng thay cho đối tượng DataSet để đọc dữ liệu nhưng không thể làm thay đổi các dòng trong cơ sở dữ liệu. Đối tượng DataSet cho phép lưu một bản sao của các dòng trong cơ sở dữ liệu. Vì thế, ta có thể xử lý dữ liệu trên bảng sao mà không cần giữ kết nối.

4.1.1. Lớp SqlCommand

Đối tượng SqlCommand được sử dụng để thực thi một lệnh truy vấn nào đó tới cơ sở dữ liệu SQL Server. Tương tự, các đối tượng OleDbCommand và OdbcCommand

dùng để thực thi một truy vấn trên cơ sở dữ liệu có hỗ trợ OLE DB (như Access hay Oracle) và ODBC.

Có hai cách để tạo một đối tượng SqlCommand: Dùng phương thức khởi tạo của lớp SqlCommand hoặc gọi phương thức CreateCommand() của đối tượng SqlConnection. Chúng ta cũng có thể dùng cùng một cách cho đối tượng OleDbCommand hay OdbcCommand.

- Tạo đối tượng SqlCommand dùng phương thức khởi tạo: Lớp SqlCommand có 4 phương thức tạo lập sau:

```
SqlCommand()
SqlCommand(string commandText)
SqlCommand(string commandText, SqlConnection mySqlConnection)
SqlCommand(string commandText, SqlConnection mySqlConnection,
           SqlTransaction mySqlTransaction)
```

Trong đó: commandText: chứa lệnh truy vấn SQL, tên thủ tục hay tên bảng muốn lấy dữ liệu; mySqlConnection: Là đối tượng kết nối, phương tiện liên lạc với cơ sở dữ liệu; mySqlTransaction: Là đối tượng giao dịch cơ sở dữ liệu SQL.

Trước khi sử dụng đối tượng SqlCommand, ta cần tạo ra một kết nối để liên lạc với cơ sở dữ liệu SQL Server.

```
// create a SqlConnection object to connect to the database
sqlConnection.ConnectionString = "server=.;  
database=RestaurantManagement; Integrated Security=true;");
```

Tiếp theo, tạo một đối tượng lệnh SqlCommand và gắn với kết nối

```
SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Connection = sqlConnection;
```

Đối tượng SqlCommand sẽ phải sử dụng một đối tượng SqlConnection để liên lạc với cơ sở dữ liệu. Thuộc tính CommandType của đối tượng Command xác định cách mà lệnh truy vấn được thực thi. Thuộc tính này nhận các giá trị trong enum CommandType (System.Data.CommandType).

Chuỗi lệnh truy vấn cần thực thi được gửi vào đối tượng Command qua thuộc tính CommandText. Ví dụ sau gán một lệnh SELECT cho đối tượng Command để lấy ra năm khách hàng đầu tiên trong bảng Food:

```
sqlCommand.CommandText = "Select top 5 Name, Unit, Price from Food";
```

Cũng có thể gửi lệnh truy vấn và đối tượng kết nối vào đối tượng Command cùng một lúc khi tạo đối tượng Command như sau:

```
SqlCommand mySqlCommand = new SqlCommand("Select top 5 Name, Unit,
Price from Food");
```

- Tạo đối tượng SqlCommand bằng phương thức CreateCommand: Thay vì tạo đối tượng SqlCommand dùng phương thức khởi tạo, ta dùng phương thức CreateCommand() của đối tượng SqlConnection. Phương thức này trả về một đối tượng SqlCommand mới. Ví dụ:

```
SqlCommand sqlCommand = sqlConnection.CreateCommand();
```

Đối tượng sqlCommand sẽ sử dụng sqlConnection để liên lạc với cơ sở dữ liệu.

4.1.2. Thực thi các lệnh SELECT và TableDirect

Lệnh TableDirect thực ra chỉ là một lệnh SELECT trả về tất cả các dòng, các cột của một bảng nào đó. Đối tượng Command có ba phương thức được dùng để thực thi một câu lệnh SELECT hay TableDirect (Bảng 4.1)

Bảng 4.1. Những phương thức lấy thông tin từ cơ sở dữ liệu

Phương thức	Kiểu trả về	Mô tả
ExecuteReader()	SqlDataReader	Được dùng để thực thi các lệnh SELECT, TableDirect hay lời gọi thủ tục có trả về một tập dữ liệu. Kết quả trả về này được lưu trong một đối tượng DataReader.
ExecuteScalar()	object	Dùng để thực thi các lệnh SELECT chỉ trả về một giá trị (những giá trị khác sẽ bị bỏ qua). Kết quả trả về được lưu trong một object.
ExecuteXmlReader()	XmlReader	Được dùng để thực thi các lệnh SELECT và trả về dữ liệu XML. Kết quả trả về được lưu trong một đối tượng XmlReader. Phương thức này chỉ áp dụng cho lớp SqlCommand.

- Phương thức ExecuteReader:* Phương thức này thực thi một lệnh SELECT và kết quả trả về được lưu trong một đối tượng DataReader. Sau đó, bạn có thể dùng đối tượng này để đọc các dòng trả về từ cơ sở dữ liệu. Ví dụ: Đoạn mã sau tạo các đối tượng cần thiết và thực thi một câu lệnh SELECT để lấy ra 5 dòng đầu tiên trong bảng Customers.

```
SqlConnection sqlConnection =
    new SqlConnection("server=.;"
                    database=RestaurantManagement; Integrated Security=true; ")
SqlCommand sqlCommand = sqlConnection.CreateCommand();
sqlCommand.CommandText =
    " Select top 5 Name, Unit, Price from Food";
sqlConnection.Open();
SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();
```

Lưu ý: Phương thức Open() của đối tượng SqlConnection chỉ được gọi ngay trước khi gọi hàm thực thi một truy vấn (ở đây là phương thức ExecuteReader của đối tượng SqlCommand). Bằng cách này, có thể giảm tối đa chi phí thời gian kết nối cơ sở dữ liệu và do đó tối ưu hóa được các tài nguyên cơ sở dữ liệu.

Tập kết quả trả về bởi đối tượng SqlCommand được lưu trong một đối tượng SqlDataReader có tên là SqlDataReader. Có thể đọc lần lượt các dòng từ SqlDataReader bằng phương thức Read(). Phương thức Read() trả về một giá trị Boolean. Giá trị đúng (*true*) có nghĩa là còn một dòng khác trong tập kết quả chưa được đọc. Giá trị sai (*false*) có nghĩa là tất cả các dòng đã được đọc.

Ta cũng có thể đọc giá trị trên một cột cụ thể của một dòng từ đối tượng DataReader bằng cách đặt tên cột hoặc một số nguyên dương (chỉ ra thứ tự tương ứng của cột đó trong tập các cột, giá trị 0 tương ứng với cột đầu tiên) trong cặp dấu ngoặc vuông [] ngay sau đối tượng DataReader. Chẳng hạn, để đọc giá trị trên cột Name của dòng hiện tại, ta viết SqlDataReader[“Name”] hoặc SqlDataReader[0].

Vì mỗi lần, phương thức Read() chỉ đọc được một dòng nên để đọc hết tất cả các dòng, ta dùng một vòng lặp để đọc từng dòng như sau:

```
while (sqlDataReader.Read())
{
    Console.WriteLine("Ten mon = " + sqlDataReader["Name"]);
    Console.WriteLine("Don Vi Tinh = " + sqlDataReader["Unit"]);
    Console.WriteLine("Don Gia = " + sqlDataReader["Price"]);
}
```

Sau khi sử dụng, chúng ta cần phải gọi phương thức Close() của đối tượng SqlDataReader và SqlConnection như sau:

```
// close the SqlDataReader object using the Close() method
sqlDataReader.Close();
// close the SqlConnection object using the Close() method
sqlConnection.Close();
```

Kết quả hiển thị của chương trình trong ví dụ trên sẽ là:

```
Ten mon = Bia Heniken
Don Vi Tinh = Chai
Don Gia = 20000
```

- *Thực thi lệnh SELECT dùng phương thức ExecuteScalar:* Phương thức *ExecuteScalar* được dùng để thực thi một lệnh SQL *SELECT* và chỉ trả về một giá trị, những giá trị khác được loại bỏ. Giá trị này nằm ở hàng đầu tiên, cột đầu tiên trong bảng kết quả và có kiểu object.

Ví dụ: Sử dụng phương thức ExecuteScalar để thực thi một câu lệnh SELECT đếm (COUNT) số dòng của bảng. Trong ví dụ này, ta gán một lệnh SELECT có sử dụng hàm COUNT() cho thuộc tính CommandText của đối tượng SqlCommand. Lệnh SELECT này sẽ trả về số dòng của bảng Products trong cơ sở dữ liệu RestaurantManagement.

```
mySqlCommand.CommandText = "SELECT COUNT(*)from Food";
```

Tiếp theo, thực thi câu lệnh SELECT bằng cách gọi phương thức ExecuteScalar.

```
int returnValue = (int) sqlCommand.ExecuteScalar();
```

Vì kết quả trả về có kiểu object nên ta phải ép kiểu về int trước khi lưu vào biến `returnValue`. Đoạn mã trong Chương trình 4.1 minh họa chương trình hoàn chỉnh để đếm số dòng của bảng Food.

Chương trình 4.1. Thực thi lệnh SELECT dùng phương thức ExecuteScalar

```
SqlConnection sqlConnection = new SqlConnection(  
    "server=.; database=RestaurantManagement; Integrated Security=true");  
  
SqlCommand sqlCommand = sqlConnection.CreateCommand();  
sqlCommand.CommandText = "Select count(*) from Food";  
sqlConnection.Open();  
  
// call the ExecuteScalar() method of the SqlCommand object  
// to run the SELECT statement  
int returnValue = (int) sqlCommand.ExecuteScalar();  
Console.WriteLine("So luong mon an nha hang hien co la: {0} mon",  
    returnValue);  
sqlConnection.Close();
```

Kết quả trả về của chương trình phụ thuộc vào số dòng trong bảng Food của cơ sở dữ liệu RestaurantManagement:

```
So luong mon an nha hang hien co la: 12 mon
```

4.1.3. Thực thi các lệnh làm thay đổi thông tin trong cơ sở dữ liệu

Đối tượng `Command` cung cấp phương thức `ExecuteNonQuery` cho phép thực thi các lệnh không yêu cầu dữ liệu trả về từ cơ sở dữ liệu. Phần này trình bày cách sử dụng phương thức `ExecuteNonQuery` để thực thi các truy vấn để thêm (INSERT), xóa (DELETE) hay cập nhật (UPDATE) dữ liệu trong các bảng.

Bạn cũng có thể dùng phương thức `ExecuteNonQuery` để gọi một thủ tục (Stored Procedure) hay thực thi một lệnh DDL (Data Definition Language) như tạo bảng (CREATE TABLE), tạo chỉ mục (CREATE INDEX). Phương thức `ExecuteNonQuery` có kiểu trả về là `int`, được dùng để thực thi các lệnh SQL không trả về tập kết quả như INSERT, UPDATE, DELETE, các lệnh DDL, lời gọi thủ tục không trả về dữ liệu. Giá trị trả về của hàm này chính là số dòng bị ảnh hưởng khi thực thi truy vấn.

4.1.4. Thực thi các lệnh INSERT, UPDATE và DELETE

- *Thêm dòng mới bằng lệnh INSERT:* Để thực thi một lệnh INSERT bằng cách dùng phương thức `ExecuteNonQuery`, trước hết, cần phải tạo một đối tượng `Command`:

```
SqlCommand sqlCommand = sqlConnection.CreateCommand();
```

Tiếp theo, gán chuỗi lệnh INSERT cho thuộc tính `CommandText` của đối tượng `Command`. Ví dụ sau gán một lệnh INSERT cho thuộc tính `CommandText` của đối tượng `SqlCommand` để thêm một dòng mới vào bảng Food:

```
sqlCommand.CommandText = "INSERT INTO Food (" + " Name, Unit, FoodCategoryID, Price, Notes )" + "VALUES ('Khoai tay chien', 'Dia', 1, 25000, NULL );"
```

Cuối cùng, gọi phương thức ExecuteNonQuery để thực thi lệnh INSERT.

```
int numberOfRows = sqlCommand.ExecuteNonQuery();
```

Phương thức ExecuteNonQuery trả về một giá trị nguyên (kiểu int) cho biết số dòng bị ảnh hưởng khi lệnh được thực thi. Trong ví dụ này, giá trị trả về chính là số dòng được thêm vào bảng Food. Chỉ có một hàng được thêm vào bởi lệnh INSERT, do đó kết quả trả về là 1.

- *Cập nhật dữ liệu bằng lệnh UPDATE:* Để cập nhật dữ liệu cho dòng mới được thêm vào ở phần trước, ta dùng lệnh UPDATE. Đoạn mã sau gán chuỗi lệnh UPDATE cho thuộc tính CommandText của đối tượng Command để thay đổi dữ liệu trong cột CompanyName của dòng vừa được thêm vào. Sau đó, gọi phương thức ExecuteNonQuery để thực thi lệnh UPDATE:

```
sqlCommand.CommandText = "UPDATE Food SET Price = 28000 " + " WHERE ID = 13";  
numberOfRows = sqlCommand.ExecuteNonQuery();
```

Phương thức ExecuteNonQuery trả về số dòng bị thay đổi bởi lệnh UPDATE. Trong trường hợp này, giá trị trả về là 1 vì chỉ có 1 dòng bị thay đổi.

- *Xóa dữ liệu bằng lệnh DELETE:* Cuối cùng, để xóa dữ liệu, ta dùng lệnh DELETE như ví dụ sau:

```
sqlCommand.CommandText = "DELETE FROM Food WHERE ID = 13";  
numberOfRows = sqlCommand.ExecuteNonQuery();
```

Kết quả trả về của phương thức ExecuteNonQuery là 1 vì chỉ có một dòng bị xóa bởi lệnh DELETE.

4.1.5. Các giao dịch trong cơ sở dữ liệu

Các lệnh SQL có thể được nhóm với nhau để tạo thành các giao dịch (*transaction*). Giao dịch có thể được thực hiện thành công (*commit*) hoặc bị gỡ bỏ (*roll back*) như một lệnh thông thường. Chẳng hạn, trong giao dịch ngân hàng, bạn có thể rút tiền từ một tài khoản vào chuyển nó sang một tài khoản khác. Trong trường hợp này, hoặc cả hai thay đổi phải được thực hiện thành công (*commit*) hoặc phải được gỡ bỏ nếu một trong hai thao tác xảy ra lỗi (*roll back*). Trong phần này, ta sẽ làm quen với việc sử dụng giao dịch cơ sở dữ liệu với ADO .Net.

Có 3 lớp giao dịch (Transaction): SqlTransaction, OleDbTransaction và OdbcTransaction. Các đối tượng thuộc lớp Transaction được dùng để biểu diễn một giao dịch trong ADO .Net.

Ta xét một ví dụ về giao dịch gồm hai câu lệnh INSERT. Lệnh INSERT thứ nhất sẽ thêm một dòng vào bảng Category còn lệnh INSERT thứ hai thêm một dòng vào bảng Food. Dòng mới trong bảng Food sẽ tham chiếu đến dòng mới trong bảng Category (FoodCategoryID của Food bằng ID mới được thêm của bảng Category, trường hợp này là ID = 2). Hai lệnh INSERT này được viết như sau:

```
INSERT INTO Category ( Name, Type) VALUES ( 'Hai san', 1)  
INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes)  
VALUES ('Khoai tay chien', 'Dia', 2, 25000, NULL )"
```

Để thực thi một giao dịch dùng *SqlConnection*, thực hiện theo các bước sau:

- **B1:** Tạo một đối tượng *SqlTransaction* và bắt đầu giao dịch bằng cách gọi phương thức *BeginTransaction* của đối tượng *SqlConnection*;
- **B2:** Tạo đối tượng *SqlCommand* để thực thi truy vấn SQL;
- **B3:** Gán đối tượng *SqlTransaction* đã tạo ở Bước 1 cho thuộc tính *Transaction* của đối tượng *SqlCommand*;
- **B4:** Gán câu lệnh *INSERT* thứ nhất cho thuộc tính *CommandText* của đối tượng *SqlCommand*. Lệnh *INSERT* này thêm một dòng mới vào bảng *Food*;
- **B5:** Thực thi truy vấn bằng cách gọi hàm *ExecuteNonQuery* của đối tượng *SqlCommand*. Ta dùng phương thức này vì lệnh *INSERT* không trả về tập dữ liệu;
- **B6:** Gán câu lệnh *INSERT* thứ hai cho thuộc tính *CommandText* của đối tượng *SqlCommand*. Lệnh này thêm một dòng mới vào bảng *Food*.
- **B7:** Thực thi lệnh truy vấn bằng cách gọi hàm *ExecuteNonQuery* của đối tượng *SqlCommand*.
- **B8:** Thực thi (*commit*) giao dịch bằng cách gọi phương thức *Commit* của đối tượng *SqlTransaction*. Việc này làm cho cả hai dòng mới được thêm vào cơ sở dữ liệu bởi lệnh *INSERT*.

Nếu muốn hủy bỏ các lệnh đã tạo trong giao dịch, ta gọi phương thức *Rollback* thay vì *Commit*. Theo mặc định, các giao dịch sẽ bị gỡ bỏ (*roll back*). Vì thế, luôn phải gọi phương thức *Commit* hoặc *Rollback* để chỉ rõ muốn xác nhận hay hủy bỏ các giao dịch đã tạo.

4.1.6. Truyền tham số vào các lệnh

Trong các ví dụ trước đây, giá trị được truyền vào mỗi cột được gán cứng (*hard-code*) trong các lệnh SQL. Chẳng hạn như lệnh *INSERT* sau:

```
INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes)  
VALUES ('Khoai tay chien', 'Dia', 1, 25000, NULL)
```

Giá trị của các cột Name, Unit, FoodCategoryID, Price, Notes tương ứng đều được gán cứng là 'Khoai tay chien', 'Dia', 1, 25000, NULL. Nếu phải thực thi nhiều lệnh

với giá trị các cột được gắn cứng sẽ gây nhiều rắc rối và không hiệu quả. Để giải quyết vấn đề này, ADO .Net cho phép truyền giá trị vào lệnh bằng cách dùng các tham số. Các tham số cho phép bạn truyền các giá trị khác nhau vào cùng một lệnh khi chạy chương trình.

Để thực thi một lệnh có chứa tham số, thực hiện các thao tác như sau: Tạo một đối tượng Command chứa lệnh SQL với các điểm đánh dấu tham số (parameter placeholder). Điểm đánh dấu tham số chỉ ra vị trí mà một tham số được cung cấp; Thêm các tham số vào đối tượng Command; Gán giá trị cho các tham số; và cuối cùng là thực thi lệnh.

- *Bước 1:* Tạo đối tượng Command chứa lệnh SQL có tham số

Điều này có thể hiểu như sau: Thay vì đưa một giá trị vào lệnh SQL, ta thay giá trị đó bằng một điểm đánh dấu tham số. Một điểm như vậy đánh dấu vị trí mà ta sẽ đưa giá trị vào đó. Cú pháp sử dụng điểm đánh dấu tùy thuộc vào cơ sở dữ liệu đang sử dụng. Với SQL Server, điểm đánh dấu được ký hiệu bởi một tên, bắt đầu bằng dấu @. Chẳng hạn như @Name, @Price. Câu lệnh INSERT sau sử dụng 5 điểm đánh dấu tham số tương ứng với giá trị trên 5 cột Name, Unit, FoodCategoryID, Price, Notes của bảng Food.

```
INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes)
VALUES ( @Name, @Unit, @FoodCategoryID, @Price, @Notes)
```

Điểm đánh dấu có thể được dùng thay cho giá trị của một cột bất kỳ trong các lệnh SELECT, INSERT, UPDATE hay DELETE. Sau đây là một ví dụ về các lệnh SELECT, UPDATE và DELETE có chứa điểm đánh dấu.

```
SELECT *
FROM Food
WHERE FoodCategoryID = @FoodCategoryID

UPDATE Food
SET Name = @Name
WHERE ID = @ID

DELETE FROM Food
WHERE ID = @ID
```

Để thực thi các lệnh dạng này, trước hết, ta tạo một đối tượng SqlCommand và gán lệnh truy vấn cho thuộc tính CommandText của SqlCommand.

```
SqlCommand sqlCommand = sqlConnection.CreateCommand();
sqlCommand.CommandText =
    "INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes) "
    + "VALUES (@Name, @Unit, @FoodCategoryID, @Price, @Notes);"
```

Lệnh INSERT được dùng để thêm một hàng vào bảng Food. Giá trị của các cột trên dòng này sẽ được truyền vào lệnh qua các tham số. Trước khi thực thi lệnh, cần phải thêm các tham số thực sự vào đối tượng SqlCommand.

- *Bước 2: Thêm các tham số vào đối tượng Command*

Để thêm các tham số thực vào đối tượng Command, ta dùng phương thức Add. Phương thức này có nhiều dạng. Phần này trình bày một dạng của phương thức Add với 5 tham số: Tên tham số hay tên dùng để ký hiệu điểm đánh dấu tham số trong lệnh SQL. Chẳng hạn, @Name là tên tham số đầu tiên trong lệnh INSERT ở ví dụ trên; Kiểu dữ liệu của cột trong cơ sở dữ liệu. với SQL Server, các kiểu này được định nghĩa bởi kiểu liệt kê System.Data.SqlDbType; Kích thước tối đa của giá trị tham số. Tham số này chỉ được dùng cho các kiểu dữ liệu có chiều dài thay đổi như char, varchar hay nvarchar.

Trong bước 1, thuộc tính CommandText của SqlCommand được gán lệnh với năm điểm đánh dấu tham số:

```
sqlCommand.CommandText =
    "INSERT INTO Food (" +
    " Name, Unit, FoodCategoryID, Price, Notes" +
    ") VALUES (" +
    " @Name, @Unit, @FoodCategoryID, @Price, @Notes " +
    ")";
```

Đoạn mã sau dùng phương thức Add để thêm năm tham số vào đối tượng SqlCommand:

```
sqlCommand.Parameters.Add("@Name ", SqlDbType.NVarChar, 1000);
sqlCommand.Parameters.Add("@Unit ", SqlDbType.NVarChar, 100);
sqlCommand.Parameters.Add("@FoodCategoryID ", SqlDbType.Int);
sqlCommand.Parameters.Add("@Price ", SqlDbType.Int);
sqlCommand.Parameters.Add("@Notes ", SqlDbType.NVarChar, 3000);
```

Phương thức Add được gọi qua thuộc tính Parameters của đối tượng SqlCommand. Một đối tượng SqlCommand chứa thuộc tính Parameters có kiểu SqlParameterCollection để lưu trữ nhiều tham số. Mỗi tham số được lưu trong một đối tượng SqlParameter. Lớp SqlParameterCollection cung cấp phương thức Add để thêm một đối tượng SqlParameter. Vì thế, để thêm một tham số vào SqlCommand, ta gọi phương thức Add của thuộc tính Parameters.

Trong đoạn mã trên, tham số @Name, @Unit, @Notes được định nghĩa là một chuỗi ký tự Unicode kiểu NVarChar với chiều dài chuỗi là 1000, 100 và 3000 (nghĩa là tham số này nhận giá trị là một chuỗi có tối đa 1000, 100 hoặc 3000 ký tự). Tương tự, các tham số @FoodCategoryID và @Price được định nghĩa là kiểu số nguyên.

- *Bước 3: Truyền giá trị cho các tham số*

Giá trị của mỗi tham số được gán qua thuộc tính Value. Để truy xuất đến một tham số trong Command, ta đặt tên tham số trong cặp dấu mốc vuông [] ngay sau thuộc tính Parameters của đối tượng Command. Đoạn mã sau minh họa cách gán giá trị cho các tham số:

```
sqlCommand.Parameters["@Name"].Value = "Khoai tay chien";
```

```
sqlCommand.Parameters["@Unit"].Value = "Dia";
mySqlCommand.Parameters["@FoodCategoryID"].Value = 2;
sqlCommand.Parameters["@Price"].Value = 25000;
```

Trong ví dụ này, giá trị của các tham số @Name, @Unit, @FoodCategoryID và @Price được gán lần lượt là Khoai tay chien, Dia, 2 và 25000. Những giá trị này sẽ được thay thế vào các điểm đánh dấu của lệnh INSERT

```
INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes)
VALUES (@Name, @Unit, @FoodCategoryID, @Price, @Notes)
```

Để đơn giản, ta cũng có thể vừa thêm một tham số, vừa gán giá trị cho nó như ví dụ sau:

```
sqlCommand.Parameters.Add("@Name",
SqlDbType.NVarChar, 1000).Value = "Khoai tay chien";
```

Đối với những cột có thể nhận giá trị null, ta có thể gán giá trị null bằng cách đặt thuộc tính IsNullable của tham số là true hoặc gán cho thuộc tính Value một giá trị đặc biệt DBNull.Value. Giá trị null thể hiện cho một giá trị chưa biết. Trong trường hợp này, giá trị của tham số sẽ tự động được chuyển sang giá trị NULL trong cơ sở dữ liệu khi thay thế vào trong lệnh truy vấn

```
sqlCommand.Parameters["@Notes"].IsNullable = true;
```

hoặc

```
mySqlCommand.Parameters["@Notes"].Value = DBNull.Value;
```

- *Bước 4: Thực thi lệnh*

Để thực thi lệnh, ta sử dụng các phương thức thực thi của đối tượng Command như ExecuteNonQuery, ExecuteScalar...

Nếu muốn thực thi các lệnh INSERT, UPDATE hay DELETE, ta dùng phương thức ExecuteNonQuery. Nếu muốn thực thi lệnh SELECT, dùng các phương thức ExecuteReader, ExecuteScalar hoặc ExecuteXmlReader.

4.1.7. Gọi các thủ tục SQL Server

Như đã trình bày trong những phần trước, hằng số CommandType.StoredProcedure được gán cho thuộc tính CommandType của đối tượng Command cho biết lệnh cần được thực thi là một thủ tục SQL (*Stored Procedure*). Tuy nhiên, cách thực sự không hiệu quả bằng cách dùng lệnh T-SQL EXECUTE. Nếu dùng lệnh T-SQL EXECUTE, ta có thể đọc các giá trị trả về bởi lệnh RETURN từ Stored Procedure. Điều này không thể thực hiện được bằng cách thiết lập giá trị CommandType.StoredProcedure cho thuộc tính CommandType.

Có hai cách để thực thi một thủ tục (*stored procedure*) phụ thuộc vào việc thủ tục có trả về tập dữ liệu hay không. Tập dữ liệu kết quả (*result set*) là một hay nhiều dòng trả về từ một bảng bởi câu lệnh SELECT.

Cách 1:

- *Thực thi một Stored Procedure không trả về dữ liệu:* Nếu một thủ tục không trả về dữ liệu, việc thực thi nó được thực hiện qua các bước sau:

B1. Tạo một đối tượng Command và gán giá trị cho thuộc tính CommandText là lệnh EXECUTE theo sau bởi tên thủ tục cần gọi.

B2. Thêm các tham số cần thiết cho lời gọi thủ tục vào đối tượng Command. Những tham số nào lưu giá trị trả về phải được thiết lập thuộc tính Direction là ParameterDirection.Output. Những tham số dạng này được định nghĩa bởi từ khóa OUTPUT trong lời gọi thủ tục hoặc được trả về bởi lệnh RETURN trong thủ tục.

B3. Thực thi lệnh bằng cách gọi phương thức ExecuteNonQuery của đối tượng Command.

B4. Đọc giá trị từ các tham số.

- *Lấy dữ liệu trả về từ tham số dùng từ khóa OUTPUT*

Đoạn mã SQL sau định nghĩa một thủ tục có tên InsertFood. Thủ tục này được dùng để thêm một dòng vào bảng Food, sau đó trả về giá trị trên cột ID của dòng mới qua một tham số ra (*OUTPUT Parameter*). Tham số này có tên là @ID

```
CREATE PROCEDURE InsertFood
    @ID int output,
    @Name nvarchar(3000),
    @Unit nvarchar(3000),
    @FoodCategoryID int,
    @Price int,
    @Notes nvarchar(3000)
AS
    INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes)
    VALUES ( @Name, @Unit, @FoodCategoryID, @Price, @Notes)
    SELECT @ID = SCOPE_IDENTITY()
```

Bước 1: Tạo đối tượng Command và thiết lập lệnh truy vấn dùng EXECUTE

Tạo một đối tượng Command, gán giá trị cho thuộc tính CommandText của nó là lệnh truy vấn EXECUTE theo sau bởi lời gọi thủ tục và danh sách các tên tham số.

```
SqlCommand sqlCommand = sqlConnection.CreateCommand();
sqlCommand.CommandText =
"EXECUTE InsertFood @ID OUTPUT, @Name, @Unit, " +
"@FoodCategoryID, @Price, @Notes";
```

Trong đoạn mã trên, tham số @ID theo sau bởi từ khóa OUTPUT cho biết đây là tham số có thể lưu lại kết quả sau khi gọi thủ tục. Các tham số khác chứa các giá trị để thêm một dòng mới bởi lệnh INSERT.

Bước 2: Thêm các tham số cần thiết vào đối tượng Command

Ở bước này, ta tiến hành thêm các tham số vào đối tượng Command. Lưu ý: phải đặt giá trị ParameterDirection.Output cho thuộc tính Direction của các tham số ra. Chẳng hạn, thủ tục InsertFood có một tham số ra để lưu ID của dòng mới. Do đó, đối tượng Command yêu cầu một tham số với Direction là Output.

```
sqlCommand.Parameters.Add("@ID", SqlDbType.Int);
sqlCommand.Parameters["@ID"].Direction = ParameterDirection.Output;
```

Các tham số khác được truyền vào lời gọi thủ tục như sau:

```
sqlCommand.Parameters.Add("@Name", SqlDbType.NVarChar, 1000).Value =
"Tom hap bia";
sqlCommand.Parameters.Add("@Unit", SqlDbType.NVarChar, 100).Value =
"Dia";
sqlCommand.Parameters.Add("@FoodCategoryID", SqlDbType.Int).Value=2;
sqlCommand.Parameters.Add("@Price", SqlDbType.Int).Value = 100000;
sqlCommand.Parameters.Add("@Notes", SqlDbType.NVarChar, 3000).Value =
DBNull.Value;
```

Lưu ý: kiểu dữ liệu SqlDbType của tham số phải tương ứng với kiểu của tham số đã khai báo khi tạo thủ tục.

Bước 3: Thực thi lệnh bởi phương thức ExecuteNonQuery

Để thực hiện việc gọi thủ tục với giá trị các tham số đã truyền vào, gọi phương thức ExecuteNonQuery của đối tượng Command.

```
sqlCommand.ExecuteNonQuery();
```

Bước 4: Đọc kết quả từ tham số

Bước cuối cùng là đọc các giá trị trả về được lưu trong các tham số ra. Để lấy được dữ liệu từ những tham số này, ta dùng thuộc tính Value. Chẳng hạn, đoạn mã sau dùng để hiển thị giá trị của tham số ProductID sinh ra bởi lệnh thêm một dòng mới vào bảng Products.

```
Console.WriteLine("Mon an moi them vao co ID = "+
sqlCommand.Parameters["@ID"].Value);
```

Kết quả chương trình sẽ là:

```
Mon an moi them vao co ID = 19
```

- *Lấy dữ liệu trả về bởi lệnh RETURN*

Đoạn mã SQL sau định nghĩa một thủ tục có tên AddProduct2 tương tự như thủ tục AddProduct nhưng dùng lệnh RETURN để trả về ProductID của dòng mới thay vì dùng tham số OUTPUT.

```
CREATE PROCEDURE InsertFood2
    @Name nvarchar(3000),
    @Unit nvarchar(3000),
    @FoodCategoryID int,
    @Price int,
    @Notes nvarchar(3000)
AS
    DECLARE @ID int
    INSERT INTO Food (Name, Unit, FoodCategoryID, Price, Notes)
    VALUES ( @Name, @Unit, @FoodCategoryID, @Price, @Notes)
    SELECT @ID = SCOPE_IDENTITY()
RETURN @ID
```

Lệnh *RETURN* được đặt cuối cùng để trả về ID của dòng mới được thêm vào bảng Food. Vì thủ tục này không trả về một tập các dòng nên ta cũng áp dụng bốn bước đã trình bày ở phần trước để thực thi lời gọi thủ tục bằng ADO .NET. Điểm khác biệt duy nhất cấu trúc của lệnh EXECUTE được gán cho thuộc tính CommandText trong bước 1. Lệnh EXECUTE trong trường hợp này được viết như sau:

```
sqlCommand.CommandText =
    "EXECUTE @ID = InsertFood2 @Name, @Unit, "
    "@FoodCategoryID, @Price, @Notes";
```

Chú ý sự thay đổi vị trí của tham số @ID. Nó được chuyển về nằm ngay sau từ khóa EXECUTE, theo sau bởi dấu bằng “=” và tên của thủ tục. Sự thay đổi này là cần thiết vì thủ tục InsertFood2 dùng câu lệnh RETURN để trả về giá trị ID và gắn cho tham số @ID.

Cách 2: Thực thi một Stored Procedure có trả về tập dữ liệu

Nếu thủ tục có trả về một tập dữ liệu, thực hiện theo các bước sau:

B1. Tạo một đối tượng Command và gán lệnh EXECUTE theo sau bởi tên thủ tục cho thuộc tính CommandText.

B2. Thêm các tham số cần thiết vào đối tượng Command. Nhớ gán giá trị ParameterDirection.Output cho thuộc tính Direction của các tham số nhận giá trị trả về (OUTPUT parameter).

B3. Thực thi lệnh dùng phương thức ExecuteReader, lưu kết quả trả về vào đối tượng DataReader.

B4. Đọc các dòng trong tập kết quả bằng phương thức Read của đối tượng DataReader

B5. Đóng đối tượng DataReader. Việc này phải được thực hiện trước khi đọc giá trị các tham số OUTPUT.

B6. Đọc giá trị của các tham số nhận giá trị trả về.

4.2. Thao tác cơ sở dữ liệu với lớp SqlDataReader

Đối tượng SqlDataReader được dùng để đọc các dòng lấy được từ cơ sở dữ liệu SQL Server. Tương tự, các đối tượng OleDbDataReader, OdbcDataReader dùng để đọc dữ liệu từ các cơ sở dữ liệu hỗ trợ OLE DB, ODBC.

4.2.1. Tạo đối tượng SqlDataReader

Đối tượng DataReader chỉ có thể được tạo bằng cách gọi phương thức ExecuteReader của đối tượng Command.

Ví dụ: Đoạn mã sau tạo các đối tượng cần thiết và thực thi một câu lệnh SELECT lấy về 5 dòng đầu tiên trong bảng Food của cơ sở dữ liệu RestaurantManagement trong SQL Server. Sau đó, lưu kết quả vào đối tượng SqlDataReader.

```
SqlConnection sqlConnection =
    new SqlConnection("server=.;"
                     database=RestaurantManagement; Integrated Security=true;")
SqlCommand sqlCommand = sqlConnection.CreateCommand();
sqlCommand.CommandText =
    " Select top 5 Name, Unit, Price from Food";
sqlConnection.Open();
SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();
```

Trong ví dụ này, đối tượng SqlDataReader trả về bởi phương thức ExecuteReader có tên là sqlDataReader.

4.2.2. Đọc dữ liệu từ SqlDataReader

Phương thức Read() được dùng để đọc các dòng từ một đối tượng DataReader. Phương thức này trả về một giá trị Boolean. Giá trị true có nghĩa là còn một dòng khác chưa được đọc, false nếu ngược lại. Việc đọc giá trị trên một cột được thực hiện bằng cách đặt tên cột hoặc thứ tự của cột vào cặp dấu mốc vuông [] ngay sau đối tượng DataReader. Việc sử dụng thứ tự hay chỉ số cột được chương trình xử lý nhanh hơn so với cách dùng tên cột.

Chẳng hạn, để đọc giá trị trên cột Name, ta viết sqlDataReader[“Name”] hoặc sqlDataReader[0]. Vì Name là cột đầu tiên trong bảng kết quả trả về nên có thứ tự là 0. Ví dụ, hai đoạn mã sau minh họa cách lấy giá trị của một cột từ DataReader bằng cách dùng tên cột và chỉ số cột.

```
// Dùng tên cột
while (sqlDataReader.Read())
{
    Console.WriteLine("Ten mon = " + sqlDataReader["Name"]);
```

```
        Console.WriteLine("Don Vi Tinh = " + sqlDataReader["Unit"]);
        Console.WriteLine("Don Gia = " + sqlDataReader["Price"]);
    }
    // Dùng chỉ số cột
    while (sqlDataReader.Read())
    {
        Console.WriteLine("Ten mon = " + sqlDataReader[0]);
        Console.WriteLine("Don Vi Tinh = " + sqlDataReader[1]);
        Console.WriteLine("Don Gia = " + sqlDataReader[2]);
    }
}
```

Mặc dù cách dùng chỉ số cột làm cho chương trình chạy nhanh hơn nhưng lại làm cho mã khó hiểu, không linh hoạt vì phải xác định một cột có chỉ số là bao nhiêu và ngược lại. Nếu vị trí của các cột trong lệnh SELECT bị thay đổi, ta phải thay đổi chỉ số cột đã gắn cứng trong mã của chương trình. Mặt khác, việc dùng chỉ số cột cũng làm cho lập trình viên khó đọc hiểu mã hơn.

Một giải pháp để khắc phục vấn đề này là dùng phương thức GetOrdinal của đối tượng DataReader. Phương thức này trả về vị trí của cột có tên được truyền qua tham số của hàm. Vị trí này còn được gọi là thứ tự của cột (*ordinal*). Ta dùng kết quả trả về của hàm GetOrdinal làm thứ tự cột để lấy giá trị cột đó từ DataReader.

Đây là cách tốt nhất để lấy dữ liệu từ DataReader, cả về tính mềm dẻo, linh động và hiệu quả cao. Ví dụ, đoạn mã sau sử dụng phương thức GetOrdinal để lấy vị trí các cột có trong câu lệnh SELECT trên. Sau đó sử dụng chỉ số cột để lấy giá trị trên các cột này từ đối tượng DataReader.

```
int nameColPos = sqlDataReader.GetOrdinal("Name");
int unitColPos = sqlDataReader.GetOrdinal("Unit");
int priceColPos = sqlDataReader.GetOrdinal("Price");

while (sqlDataReader.Read())
{
    Console.WriteLine("Ten mon = " + sqlDataReader[nameColPos]);
    Console.WriteLine("Don Vi Tinh = " +
sqlDataReader[unitColPos]);
    Console.WriteLine("Don Gia = " + sqlDataReader[priceColPos]);
}
```

Đối tượng DataReader cần phải được đóng sau khi hoàn tất việc đọc dữ liệu bằng cách gọi phương thức Close. Lý do cần phải đóng là vì đối tượng DataReader sử dụng kết hợp với đối tượng Connection. Nếu không được đóng, kết nối vẫn được mở và dành riêng cho DataReader trong khi không còn lệnh nào được thực hiện. Một khi đã đóng DataReader, kết nối thể thê dùng để thực thi các lệnh truy vấn khác.

```
sqlDataReader.Close();
```

4.2.3. Lấy giá trị từ các cột với kiểu cụ thể

Cho đến thời điểm này, giá trị của các cột lấy được từ DataReader đều có kiểu System.Object. Mọi lớp trong C# đều được dẫn xuất từ lớp này. Trong trường hợp cần

thực hiện tính toán trên các giá trị lấy được, chúng phải được ép sang các kiểu cụ thể. Chẳng hạn, để ép đổi тип unitPrice sang kiểu decimal và nhân với 1.2, ta viết:

```
decimal newUnitPrice = (decimal)unitPrice * 1.2m;
```

Ký hiệu m nằm sau số 1.2 để cho biết đó là số ở dạng *decimal*. Việc ép một đối tượng sang kiểu nào đó là hoàn toàn có thể. Tuy nhiên, điều này không thực sự hiệu quả. Nó cũng đi ngược với các ưu điểm chính của các ngôn ngữ lập trình bậc cao: sử dụng kiểu rõ ràng. Kiểu dữ liệu rõ ràng (*strongly typing*) có nghĩa là bạn phải xác định rõ kiểu dữ liệu của biến hay đổi type lúc khai báo. Ưu điểm của kiểu dữ liệu rõ ràng là hạn chế được các lỗi trong lúc chạy chương trình do gán giá trị hay sử dụng sai kiểu dữ liệu. Trình biên dịch sẽ kiểm tra mã chương trình để bảo đảm giá trị được gán cho một biến có kiểu nào đó có hợp lệ hay không. Một quy tắc hiệu quả là cố gắng khai báo tất cả các biến, các đối tượng với kiểu thích hợp và chỉ ép kiểu khi không còn lựa chọn nào khác.

Trong ví dụ này, thay vì ép kiểu, bạn có thể dùng các phương thức *Get** của đối tượng *DataReader* có trả về kiểu thích hợp để lấy dữ liệu từ các cột. Dấu * trong *Get** có nghĩa là đối tượng *DataReader* có nhiều phương thức bắt đầu bởi *Get* và * dùng để thay thế cho tên kiểu dữ liệu trả về của phương thức đó. Chẳng hạn, phương thức *GetInt32* trả về giá trị của một cột ở dạng số nguyên 32 bit. Những phương thức dạng này đều chấp nhận tham số là tên hoặc thứ tự của cột muốn lấy dữ liệu.

```
int foodPrice = sqlDataReader.GetInt32(Price);
```

4.2.4. Sử dụng các phương thức dạng *Get** để đọc dữ liệu

Ta khảo sát một ví dụ dùng các phương thức *Get** của đối tượng *DataReader* để đọc giá trị trên các cột ID, Name của bảng Food (Bảng 4.1). Bảng sau liệt kê tên các cột, tên phương thức cần dùng và kiểu trả về khi đọc dữ liệu trên 4 cột của bảng Food

Bảng 4.1. Phương thức *Get dùng để đọc dữ liệu cho 4 cột của bảng Food**

Tên cột	Kiểu SQL Server	Phương thức <i>Get*</i>	C# return type
ID	int	GetInt32()	int
Name	nvarchar	GetString()	string

Giả sử, ta có một đối tượng *SqlDataReader* tên là *sqlDataReader* dùng để đọc giá trị trên hai cột của bảng Food. Đoạn mã sau có một vòng lặp *while* sử dụng các phương thức dạng *Get** trả về kiểu dữ liệu chuẩn của C# để lấy giá trị các cột từ đối tượng *DataReader*.

```
while (sqlDataReader.Read())
{
    int iD = sqlDataReader.GetInt32(iDColPos);
    Console.WriteLine("FoodID = " + productID);
    string name = sqlDataReader.GetString(nameColPos);
    Console.WriteLine("Food Name = " + name);
}
```

Đối tượng DataReader cung cấp hai phương thức GetFieldType và GetDataTypeName cho phép lấy tên kiểu dữ liệu chuẩn C# và tên kiểu dữ liệu SQL Server của một cột.

```
// Lấy kiểu dữ liệu chuẩn C#
Console.WriteLine("Price .NET type = " +
    sqlDataReader.GetFieldType(priceColPos));
// Kết quả trả về
Price .NET type = System.Int32
// Lấy kiểu dữ liệu CSDL SQL Server
Console.WriteLine("Price database type = " +
    sqlDataReader.GetDataTypeName(priceColPos));
// Kết quả trả về
Price database type = int
```

4.2.5. Thực thi nhiều lệnh truy vấn SQL

Thông thường, chương trình ứng dụng và cơ sở dữ liệu nằm trên hai máy tính khác nhau và liên lạc với nhau qua mạng. Mỗi khi chương trình thực thi một lệnh truy vấn, lệnh này được truyền qua mạng đến cơ sở dữ liệu và được cơ sở dữ liệu xử lý. Kết quả thực thi lại được gửi ngược về cho chương trình thông qua mạng. Như vậy, với những chương trình ứng dụng tương đối lớn, băng thông mạng sẽ bị chiếm đáng kể. Một cách hữu ích để giảm lưu lượng các gói tin trên mạng là thực hiện nhiều truy vấn SQL tại cùng một thời điểm.

Xét một ví dụ sử dụng đối tượng Command để thực thi nhiều lệnh SELECT để lấy kết quả. Đoạn mã sau tạo một đối tượng SqlCommand có tên sqlCommand và gán 2 lệnh truy vấn SELECT khác nhau cho thuộc tính CommandText

```
SqlCommand sqlCommand = sqlConnection.CreateCommand();
sqlCommand.CommandText =
    "SELECT TOP 5 Name, Unit, Price " +
    "FROM Food " +
    "ORDER BY Price;" +
    "SELECT TOP 10 InvoiceID, FoodID, Quantity " +
    "FROM BillDetails " +
    "ORDER BY InvoiceID;";
```

Lưu ý: Các lệnh SELECT được phân tách nhau bởi dấu chấm phẩy “,”.

Để chạy các lệnh SQL đã gán cho thuộc tính CommandText của đối tượng sqlCommand ở trên, ta gọi phương thức ExecuteReader để lấy về đối tượng SqlDataReader.

```
SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();
```

Lệnh này sẽ trả về 2 tập kết quả tương ứng với 2 lệnh SELECT. Để đọc dữ liệu từ tập kết quả thứ nhất, ta dùng phương thức Read() của đối tượng SqlDataAdapter. Phương thức này trả về giá trị false nếu không còn dòng nào để đọc. Khi đọc hết các dòng trong tập đầu tiên, ta gọi hàm NextResult() của đối tượng SqlDataReader để

chuyển sang đọc dữ liệu trên tập kết quả thứ 2. Phương thức NextResult cho phép DataReader chuyển sang tập kết quả tiếp theo và chỉ trả về false nếu không còn tập dữ liệu để đọc.

Đoạn mã sau minh họa cách sử dụng các phương thức Read và NextResult để đọc dữ liệu từ 2 tập kết quả của 2 lệnh SELECT.

```
do
{
    while (sqlDataReader.Read())
    {
        Console.WriteLine("sqlDataReader[0] = " +
                           sqlDataReader[0]);
        Console.WriteLine("sqlDataReader[1] = " +
                           sqlDataReader[1]);
    }
} while (sqlDataReader.NextResult());
```

Ta cũng có thể xen kẽ nhiều lệnh SELECT, INSERT, UPDATE và DELETE với nhau. Điều này giúp tiết kiệm được băng thông mạng vì nhiều lệnh chỉ được gửi một lần tới cơ sở dữ liệu

Bạn có thể sử dụng cùng một vòng lặp *do ... while* như ví dụ ở phần trước để lấy các tập dữ liệu trả về bởi câu lệnh SELECT. Thậm chí, vòng lặp này vẫn thực hiện khi không có câu lệnh SELECT trong nhóm lệnh được thực thi vì chỉ có lệnh SELECT mới trả về tập dữ liệu kết quả và phương thức NextResult chỉ trả về giá trị *true* cho lệnh SELECT, trả về giá trị *false* cho các lệnh SQL còn lại.

4.3. Lưu trữ dữ liệu với Dataset

Các đối tượng DataSet cho phép lưu trữ một bản sao thông tin của cơ sở dữ liệu. Từ đó, có thể xử lý trực tiếp trên các thông tin này trong khi kết nối đã bị ngắt. DataSet là lớp dùng chung, thuộc nhóm không kết nối. Vì thế, nó làm việc với tất cả các cơ sở dữ liệu. Không giống như DataReader, đối tượng DataSet còn cho phép đọc các dòng theo thứ tự bất kỳ và có thể thay đổi dữ liệu trong các dòng đó.

Phần này cũng trình bày cách sử dụng đối tượng DataAdapter để đọc các dòng từ cơ sở dữ liệu vào một DataSet. Lớp DataAdapter thuộc nhóm lớp kết nối và là cầu nối giữa hai nhóm lớp: kết nối và không kết nối. Có 3 lớp DataAdapter là SqlDataAdapter, OleDbDataAdapter và OdbcDataAdapter. Đối tượng DataAdapter được dùng để sao chép các dòng từ cơ sở dữ liệu vào một DataSet và cập nhật lại các thay đổi trong DataSet về cơ sở dữ liệu.

4.3.1. Lớp SqlDataAdapter

Lớp SqlDataAdapter đóng vai trò cầu nối giữa nhóm lớp kết nối và không kết nối. Nó được dùng để đóng bộ hóa dữ liệu được lưu trong một DataSet với cơ sở dữ liệu SQL Server. Để tạo một đối tượng SqlDataAdapter, ta sử dụng một trong các phương thức khởi tạo của lớp SqlDataAdapter như sau:

```
SqlDataAdapter ()  
SqlDataAdapter (SqlCommand mySqlCommand)  
SqlDataAdapter (string commandString, SqlConnection mySqlConn)  
SqlDataAdapter (string commandString, string connectionString)
```

Trong đó: *mySqlCommand*: là đối tượng SqlCommand; *commandString*: là lệnh SQL SELECT hoặc tên một thủ tục; *mySqlConn*: là đối tượng kết nối SqlConnection; *connectionString*: là chuỗi chứa thông tin kết nối cơ sở dữ liệu.

Đoạn mã sau minh họa cách tạo một đối tượng SqlDataAdapter dùng phương thức khởi tạo đầu tiên SqlDataAdapter().

```
SqlDataAdapter sqlDataAdapter = new SqlDataAdapter();
```

Trước khi sử dụng đối tượng sqlDataAdapter để thao tác với DataSet, ta phải gán một đối tượng SqlCommand chứa lệnh SELECT hoặc tên của một thủ tục cho thuộc tính SelectCommand. Đoạn mã sau minh họa cách tạo một đối tượng SqlCommand với thuộc tính CommandText là lệnh SELECT để lấy dữ liệu 5 dòng đầu tiên trên các cột Name, Unit và Price trong bảng Food của cơ sở dữ liệu RestaurantManagement. Sau đó, gán đối tượng SqlCommand này cho thuộc tính SelectCommand của SqlDataAdapter.

```
SqlCommand sqlCommand = sqlConnection.CreateCommand();  
sqlCommand.CommandText =  
    "SELECT TOP 5 Name, Unit, Price " +  
    "FROM Food " +  
    "ORDER BY Price";  
sqlDataAdapter.SelectCommand = sqlCommand;
```

4.3.2. Lớp DataSet

Đối tượng DataSet dùng để biểu diễn một bản sao thông tin được lưu trong cơ sở dữ liệu. Ta có thể tạo ra các thay đổi về mặt dữ liệu trên bản sao này mà không làm ảnh hưởng đến cơ sở dữ liệu đang tồn tại và sau đó đồng bộ hóa các thay đổi với cơ sở dữ liệu qua đối tượng DataAdapter. Một DataSet có thể biểu diễn các cấu trúc của cơ sở dữ liệu như bảng (Table), các hàng (Row) và các cột. Thậm chí là cả các ràng buộc hay quan hệ giữa các bảng. Chẳng hạn như ràng buộc khóa ngoại, tính duy nhất của dữ liệu.

- *Tạo một đối tượng DataSet*: Để tạo một đối tượng DataSet, ta dùng một trong hai phương thức khởi tạo sau:

```
DataSet() hoặc  
DataSet(string dataSetNameString)
```

Trong đó: *dataSetNameString* là tên của DataSet (kiểu chuỗi) được gán cho thuộc tính DataSetName của đối tượng DataSet cần tạo. Tên của DataSet có thể có hoặc không. Nếu có sử dụng, nên đặt cho nó một tên có ý nghĩa và mang tính gợi nhớ.

- *Đưa dữ liệu vào DataSet*: Phần này trình bày cách dùng phương thức Fill() của đối tượng DataAdapter để đưa dữ liệu trả về bởi lệnh SELECT, thủ tục hay một phần của tập dữ liệu đó vào DataSet. Hàm này trả về số nguyên chỉ ra số

dòng được đưa vào DataSet bởi đối tượng DataAdapter. Trước khi lấy được dữ liệu vào đưa vào *DataSet*, ta cần phải tạo các đối tượng *Connection*, *Command* và *DataAdapter* (xem ví dụ trên). Tiếp theo, đưa dữ liệu trả về từ truy vấn vào *DataSet* bằng cách gọi phương thức *Fill* của đối tượng *SqlDataAdapter*.

```
int numberOfRows = sqlDataAdapter.Fill(myDataSet, "Food");
```

Phương thức *Fill()* trả về số nguyên cho biết số dòng đã lấy và được đưa vào *DataSet* bởi đối tượng *DataAdapter*. Trong ví dụ này, kết quả trả về là 5.

Tham số thứ nhất của phương thức *Fill()* là *DataSet* dùng để lưu kết quả. Tham số thứ hai là một chuỗi, đây là tên mà bạn muốn gán cho đối tượng *DataTable* được tạo ra trong *DataSet*. Tên mà bạn muốn gán cho *DataTable* không nhất thiết phải trùng với tên bảng trong cơ sở dữ liệu. Bạn có thể dùng một tên bất kỳ, miễn sao nó mang tính gợi nhớ và biểu thị được nội dung trả về của truy vấn. Tuy nhiên, các *DataTable* thường dùng để chứa dữ liệu trong một bảng tương ứng của cơ sở dữ liệu, vì thế bạn nên lấy cùng tên để tiện cho việc xử lý.

Khi phương thức *Fill()* lần đầu tiên được gọi, các bước sau sẽ được thực hiện: Lệnh *SELECT* trong đối tượng *SqlCommand* sẽ được thực thi; Một đối tượng *DataTable* mới được tạo ra trong *DataSet*; Đưa dữ liệu trả về bởi lệnh *SELECT* vào *DataSet*. Sau khi gọi lệnh *Fill()*, cần phải đóng kết nối cơ sở dữ liệu để giải phóng tài nguyên hệ thống.

```
// Đóng kết nối CSDL  
mySqlConnection.Close();
```

Thực tế, phương thức *Fill()* sẽ tự mở và đóng đối tượng *Connection* nếu bạn không thực hiện điều này. Tuy nhiên, tốt hơn hết là mở và đóng kết nối một cách rõ ràng để chương trình thực thi theo đúng những gì đã vạch ra. Mặc khác, nếu phải dùng phương thức *Fill()* nhiều lần trong một đoạn mã ngắn, ta nên giữ kết nối ở trạng thái *Open* và chỉ đóng khi đã thực hiện xong tất cả.

Sau khi đưa dữ liệu vào *DataSet* (thực chất là một *DataTable*). Ta có thể đọc dữ liệu từ *DataTable* trong *DataSet* đó. Để lấy một *DataTable* từ *DataSet*, ta đặt tên của *DataTable* hoặc chỉ số của bảng trong cặp dấu ngoặc vuông [] ngay sau thuộc tính *Tables* của đối tượng *DataSet*.

```
DataTable myDataTable = myDataSet.Tables["Food"];  
DataTable myDataTable = myDataSet.Tables[0];
```

Đoạn mã sau sử dụng một vòng lặp *foreach* duyệt qua các dòng (*DataRow*) của bảng kết quả (*DataTable*) để xuất giá trị các cột ra màn hình.

```
foreach (DataRow myDataRow in myDataTable.Rows)  
{  
    Console.WriteLine("FoodName = " + myDataRow["Name"]);  
    Console.WriteLine("Unit = " + myDataRow["Unit"]);  
    Console.WriteLine("Price = " + myDataRow["Price"]);  
}
```

Thuộc tính Rows có kiểu DataRowCollection cho phép truy xuất đến tất cả các dòng hay các đối tượng DataRow được lưu trong DataTable. Để đọc dữ liệu trên các cột của DataRow, đặt tên cột hoặc chỉ số của cột trong cặp dấu ngoặc vuông [] ngay sau đối tượng DataRow. Chẳng hạn, để lấy giá trị trên cột Name, ta viết myDataRow[“Name”] hoặc myDataRow[0] vì ProductID là cột đầu tiên nên có chỉ số là 0.

Ngoài việc lấy tất cả các dòng của kết quả truy vấn như trên, các dạng khác của phương thức Fill() còn cho phép lấy một phần của tập kết quả sau khi truy vấn.

```
int Fill(DataSet myDataSet)
int Fill(DataTable myDataTable)
int Fill(DataSet myDataSet, string dataTableName)
int Fill(DataSet myDataSet, int startRow, int numRows,
string dataTableName)
```

Trong đó: *myDataSet*: là đối tượng DataSet chứa kết quả trả về sau truy vấn; *myDataTable*: là đối tượng DataTable chứa kết quả trả về sau truy vấn; *dataTableNames*: là chuỗi chứa tên của *DataTable* sẽ chứa dữ liệu kết quả; *startRow*: chỉ số của dòng bắt đầu lấy dữ liệu trong tập dữ kết quả; *numOfRows*: số dòng muốn lấy từ tập kết quả

Tập các dòng muốn lấy có chỉ số bắt đầu từ *startRow* đến *startRow + numOfRows* và được lưu vào một *DataTable*. Tất cả các phương thức *Fill()* đều trả về một số nguyên cho biết số dòng lấy được từ cơ sở dữ liệu.

4.3.3. Lớp *DataTable*

Đối tượng *DataTable* được dùng để biểu diễn một bảng hay chứa dữ liệu trả về từ một bảng trong cơ sở dữ liệu. Đối tượng *DataRow* được dùng để biểu diễn một dòng trong bảng (*DataTable*). Một *DataTable* có thể chứa nhiều đối tượng *DataRow*. Đối tượng *DataColumn* dùng để biểu diễn một cột của bảng (*DataTable*) hoặc dòng (*DataRow*). Một dòng *DataRow* hoặc *DataTable* có thể chứa nhiều đối tượng *DataColumn*.

4.4. Trích lọc và sắp xếp dữ liệu dùng *DataView*

Một *DataView* lưu các bản sao của các dòng trong một *DataTable*. Ưu điểm của *DataView* là có thể gắn (*bind*) nó cho một thành phần giao diện của Windows Form (chẳng hạn như điều khiển *DataGridView*).

Đối tượng *DataView* được dùng để xem nội dung các dòng trong một *DataTable* qua một bộ lọc. Ta cũng có thể dùng *DataView* để sắp xếp các dòng hoặc thêm dòng mới, cập nhật và xóa các dòng khỏi một *DataView*. Các thay đổi dữ liệu trên *DataView* cũng được cập nhật cho *DataRow* trong *DataTable* mà nó đọc dữ liệu. Để tạo một đối tượng *DataView*, ta dùng một trong các phương thức khởi tạo sau:

```
DataView()
DataView(DataTable myDataTable)
DataView(DataTable myDataTable, string filterExpression,
string sortExpression, DataViewRowState rowState)
```

Trong đó: *myDataTable*: là DataTable mà DataView liên kết tới. DataView sẽ đọc các dòng dữ liệu từ DataTable này. Giá trị của tham số này sẽ được gán cho thuộc tính Table của DataView; *filterExpression*: là một chuỗi chứa biểu thức điều kiện dùng để trích lọc các dòng. Giá trị của tham số này được gán cho thuộc tính RowFilter; *sortExpression*: là một chuỗi chứa biểu thức quy định các cột và cách sắp xếp trên cột đó. Giá trị tham số này được gán cho thuộc tính Sort của DataView.; *rowState*: là một điều kiện bổ sung cho bộ lọc để lấy các DataRowView có trạng thái tương ứng với rowState. Giá trị của tham số này được gán cho thuộc tính RowStateFilter của DataView.

Trước khi tạo DataView, ta phải tạo một DataTable để chứa các dòng đọc được từ cơ sở dữ liệu. Giả sử, ta đã có một DataTable với tên customersDT chứa dữ liệu lấy được từ bảng Customers. Ta tạo các biến để chứa biểu thức điều kiện trích lọc, sắp xếp và biến trạng thái RowState như sau:

```
string filterExpression = "Name like 'Tom%'";  
string sortExpression = "Price DESC";  
DataViewRowState rowStateFilter = DataViewRowState.OriginalRows;
```

Tiếp theo, sử dụng phương thức khởi tạo thứ 3 để tạo một DataView với tên customersDV.

```
DataView foodDV =  
    new DataView(foodDT, filterExpression, sortExpression,  
rowStateFilter);
```

Một cách khác để tạo DataView là sử dụng phương thức khởi tạo thứ nhất. Sau đó, gán giá trị cho các thuộc tính Table, RowFilter, Sort và RowStateFilter.

```
DataView foodDV = new DataView();  
foodDV.Table = customersDT;  
foodDV.RowFilter = filterExpression;  
foodDV.Sort = sortExpression;  
foodDV.RowStateFilter = rowStateFilter;
```

Một DataView lưu trữ các dòng trong các đối tượng DataRowView. Các DataRowView đọc dữ liệu từ các đối tượng DataRow lưu trong DataTable bên dưới. Đoạn mã sau sử dụng một vòng lặp *foreach* để hiển thị các đối tượng DataRowView trong DataView đã tạo ở trên. Trong đó, myDataRowView[count] trả về giá trị của cột có chỉ số xác định bởi biến *count*.

```
foreach (DataRowView myDataRowView in foodDV)  
{  
    for (int count = 0; count < foodDV.Table.Columns.Count; count++)  
    {  
        Console.WriteLine(myDataRowView[count]);  
    }  
    Console.WriteLine("");  
}
```

Chương trình 4.2. minh họa cách tạo một đối tượng DataView và dùng nó để trích lọc, sắp xếp các hàng trong bảng.

Chương trình 4.2. Tạo và sử dụng Data View

```
// create a SqlConnection object to connect to the database
SqlConnection sqlConnection = new SqlConnection(
    "server=.; database=RestaurantManagement; Integrated
    Security=true");

// create a SqlCommand object
SqlCommand mySqlCommand = sqlConnection.CreateCommand();

// create a SqlDataAdapter object
SqlDataAdapter myDataAdapter = new SqlDataAdapter(mySqlCommand);

// create a Dataset object
DataSet myDataSet = new DataSet();

// set the CommandText property of the SqlCommand object to
// the SELECT statement
mySqlCommand.CommandText = "Select Name, Unit, Price from Food";

// Open the connection to the database
sqlConnection.Open();

// Execute the sql command in the CommandText and save the result
// in the myDataSet dataset, table name is "Food"
int numRows = myDataAdapter.Fill(myDataSet, "Food");

// Close the connection to the database
sqlConnection.Close();

// Get table "Food" from myDataSet
DataTable foodTable = myDataSet.Tables["Food"];

// create filter and sort expression
string filterExpression = "Name like '%ng%'";
string sortExpression = "Price DESC";
DataViewRowState rowStateFilter = DataViewRowState.OriginalRows;

// Create a data view object to view the data in foodDT data table
// filter by Name (contain 'ng') and sort descending by Price
DataView foodView = new DataView(foodTable,
    filterExpression, sortExpression, rowStateFilter);

// Assign foodDT as Data Source of data grid view
dgvFoodList.DataSource = foodView;
```

Kết quả chạy chương trình như trong Hình 4.1.

	Tên món ăn	Đơn vị tính	Đơn giá
▶	Gà nướng muối ớt	Con	180000
	Càng cua hấp	Đĩa	100000
	Mực nướng太极	Đĩa	90000
	Gà nướng mật ong	Đĩa	90000
	Sò lông nướng mỡ hành	Đĩa	80000
	Cơm chiên Dương châu	Đĩa lớn	40000
	Cơm chiên Dương châu	Đĩa nhỏ	35000
	Rau muống xào tỏi	Đĩa	20000
*			

Hình 4.1. Minh họa sử dụng DataView

4.5. Kết chương

Trong chương này bạn đã học được cách truy vấn dữ liệu sử dụng SqlCommand và đọc kết quả trả về sử dụng SqlDataReader. SqlCommand cho phép thực thi nhiều truy vấn cùng lúc để tiết kiệm băng thông cho mạng và SqlDataReader cũng hỗ trợ đọc dữ liệu trên nhiều bảng kết quả trả về. Bên cạnh đó, chương này cũng trình bày cách lấy giá trị từ các cột với kiểu dữ liệu cụ thể cũng như cách hiệu quả đọc dữ liệu dùng chỉ số cột khi sử dụng DataReader. Ngoài việc sử dụng SqlDataReader để đọc dữ liệu ngay khi đang mở kết nối tới cơ sở dữ liệu, chương này còn hướng dẫn cách sử dụng Dataset để lưu trữ dữ liệu và thao tác trên dữ liệu này sau khi kết nối đã bị ngắt. Đối tượng DataSet dùng để biểu diễn một bản sao thông tin được lưu trong cơ sở dữ liệu. Ta có thể tạo ra các thay đổi về mặt dữ liệu trên bản sao này mà không làm ảnh hưởng đến cơ sở dữ liệu đang tồn tại và sau đó đồng bộ hóa các thay đổi với cơ sở dữ liệu qua đối tượng DataAdapter. Một DataSet có thể biểu diễn các cấu trúc của cơ sở dữ liệu như bảng (Table), các hàng (Row) và các cột. Thậm chí là cả các ràng buộc hay quan hệ giữa các bảng. Chẳng hạn như ràng buộc khóa ngoại, tính duy nhất của dữ liệu...Dataset cũng cho phép lưu trữ nhiều bảng kết quả trả về nếu người dùng thực thi nhiều truy vấn cùng lúc. Phần cuối chương còn giới thiệu cho bạn cách sắp xếp, trích lọc dữ liệu trong Dataset sử dụng DataView.

Bài tập

- Viết chương trình hiển thị danh sách toàn bộ món ăn của cửa hàng theo định dạng dưới đây (dùng ExecuteReader):

Tên món ăn	Đơn vị Tính	Đơn giá	Loại món ăn
Khoai tay chiên	Đĩa	25000	Khai vị
Bia Heniken	Chai	20000	Bia

2. Viết chương trình hiển thị chi tiết hóa đơn của một bàn cụ thể, tên bàn nhập từ bàn phím. Kết quả hiển thị theo dạng (dùng ExecuteReader)::

Ten mon an	Don vi Tinh	Don gia	So luong	Thanhtien
Khoai tay chien	Dia	25000	3	75000
Bia Heniken	Chai	20000	10	200000

3. Viết chương trình hiển thị tổng số tiền cần thanh toán của một bàn cụ thể, tên bàn nhập từ bàn phím (dùng ExecuteScalar)
4. Viết chương trình minh họa việc sử dụng phương thức Insert, Update và Delete cho bảng Food, các tham số nhập từ bàn phím (dùng ExecuteNonQuery)
- Sử dụng câu truy vấn
 - Gọi thủ tục, trả về ID trong trường hợp Insert, trả về toàn bộ thông tin của món ăn vừa được cập nhật.

Kết quả chương trình hiển thị như sau:

```

Number of rows added = 1
ID = 13
Ten mon: Khoai tay chien
Don Vi Tinh = Dia
Don Gia = 25000
Number of rows updated = 1
ID = 13
Ten mon Khoai tay chien
Don Vi Tinh = Dia
Don Gia = 28000
Number of rows deleted = 1

```

5. Viết chương trình có chức năng như hình dưới đây (có sử dụng nhiều câu lệnh SELECT trong SqlCommand, dataset và dataview)

The screenshot shows a Windows application window titled "Quản lý hóa đơn". At the top, there is a dropdown menu labeled "Vui lòng chọn bàn:" with the value "01". To the right of the dropdown, the text "Tổng tiền" is followed by a red-bordered box containing the value "590000". Below this, a grid displays four rows of food items. The columns are labeled: "Tên món ăn" (Food name), "Đơn vị tính" (Unit), "Số lượng" (Quantity), "Đơn giá" (Unit price), and "Thành tiền" (Total price). The items listed are: "Ếch thuỷ rán" (Fried frog) at 70000, "Cơm chiên Dương ch..." (Fried rice...) at 40000, "Càng cua hấp" (Boiled crab claws) at 100000, and "Sò lông nướng mỡ hành..." (Grilled scallops...) at 80000. The total amount is 590000.

Tên món ăn	Đơn vị tính	Số lượng	Đơn giá	Thành tiền
Ếch thuỷ rán	Đĩa	1	70000	70000
Cơm chiên Dương ch...	Đĩa lớn	2	40000	80000
Càng cua hấp	Đĩa	2	100000	200000
Sò lông nướng mỡ hành...	Đĩa	3	80000	240000

CHƯƠNG 5. CÁC THU VIỆN HỖ TRỢ LẬP TRÌNH CSDL

Chương này giới thiệu một số thư viện mã nguồn hỗ trợ việc đọc và ghi tập tin dạng văn bản, tập tin nhị phân, các tập tin bán cấu trúc như CSV, XML và các bảng tính Excel. .NET Framework đã cung cấp sẵn các thư viện hàm để làm việc với tập tin dạng văn bản và nhị phân. Tuy nhiên, đối với những tập tin bán cấu trúc như CSV hay bảng tính Excel, việc xử lý tương đối phức tạp nếu không sử dụng các thư viện hàm chuyên dụng. Cộng đồng người dùng .NET đã phát triển nhiều thư viện hàm tiện ích để giải quyết vấn đề này và phân phối dưới dạng các gói Nuget Package có thể được cài đặt vào dự án khi cần thiết. Ngoài ra, rất nhiều thư viện hỗ trợ mạnh mẽ cho việc lập trình cơ sở dữ liệu quan hệ cũng đã được phát triển trong nhiều năm qua, chẳng hạn như NHibernate, BLToolkit, LinqConnect, LINQ to SQL, Entity Framework, ... Chương này cũng giới thiệu thư viện Entity Framework được phát triển và phân phối chính thức bởi Microsoft.

Sau khi đọc xong chương này, người đọc có thể:

- Nắm vững cách đọc và ghi tập tin dạng văn bản và dạng nhị phân sử dụng các thư viện hàm được cung cấp sẵn trong .NET Framework;
- Biết cách tìm và cài đặt các gói thư viện hỗ trợ thông qua công cụ Nuget Package Manager;
- Hiểu và sử dụng được thư viện CsvHelper để đọc và tạo tập tin CSV;
- Sử dụng được các lớp xây dựng sẵn để thao tác với tập tin XML;
- Biết cách đọc và tạo bảng tính Excel với thư viện EPPlus;
- Hiểu và nắm vững cách truy vấn dữ liệu từ cơ sở dữ liệu quan hệ với Entity Framework.

5.1. Đọc và tạo tập tin dạng văn bản

Trong .NET Framework, namespace System.IO định nghĩa sẵn một tập các lớp, các cấu trúc và các giao diện để thao tác với tập tin, thư mục và các thiết bị nhập xuất. Trong đó, hai lớp File và FileInfo được dùng để làm việc với các tập tin cụ thể. Lớp File cung cấp các phương thức để tạo, xóa, cập nhật, sao chép hoặc di chuyển tập tin. Trong khi đó, lớp FileInfo cho biết thêm các thuộc tính, thông tin chi tiết hơn về một tập tin, chẳng hạn như người tạo, thời gian tạo, thời điểm cập nhật, ... Bảng 5.1 liệt kê các phương thức của lớp File mà chúng ta có thể sử dụng để đọc và ghi nội dung văn bản vào tập tin một cách nhanh chóng, chỉ với vài dòng lệnh.

Chương trình 5.1 cho thấy cách sử dụng phương thức WriteAllLines để tạo tập tin tasks.txt từ một mảng các chuỗi cho trước. Sau đó, dùng phương thức ReadAllLines() để đọc nội dung trong tập tin vừa tạo để hiển thị lên màn hình.

Bảng 5.1. Các phương thức của lớp File

Phương thức	Ý nghĩa
ReadAllLines(string path)	Mở tập tin được chỉ định bởi đường dẫn path và đọc nội dung tập tin, trả về một mảng các chuỗi, mỗi dòng tương ứng với một chuỗi. Sau đó đóng tập tin.
ReadAllText(string path)	Mở tập tin được chỉ định bởi đường dẫn path và đọc toàn bộ nội dung và trả về một chuỗi. Sau đó đóng tập tin.
WriteAllLines(string path, string[] lines)	Mở hoặc tạo tập tin tại đường dẫn path, ghi các chuỗi trong mảng lines theo từng dòng. Sau đó đóng tập tin.
WriteAllText(string path, string content)	Mở và ghi toàn bộ nội dung chuỗi content vào tập tin. Sau đó đóng tập tin.

Chương trình 5.1. Tạo và đọc nội dung tập tin dùng WriteAllLines và ReadAllLines

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("***** Simple I/O with the File Type **\n");
        string[] myTasks = { "Fix bathroom sink", "Call Dave",
            "Call Mom and Dad", "Play Xbox One"
        };
        // Write out all data to file tasks.txt in current folder.
        File.WriteAllLines(@"tasks.txt", myTasks);
        // Read it all back and print out.
        foreach (string task in File.ReadAllLines(@"tasks.txt"))
        {
            Console.WriteLine("TODO: {0}", task);
        }

        Console.ReadLine();
    }
}
```

Kết quả chạy chương trình:

```
***** Simple I/O with the File Type *****
TODO: Fix bathroom sink
TODO: Call Dave
TODO: Call Mom and Dad
TODO: Play Xbox One
[Program exited with exit code 0]
```

Việc sử dụng các phương thức trên đây tương đối đơn giản và dễ hiểu. Tuy nhiên, chúng không hiệu quả trong trường hợp tập tin cần đọc có dung lượng lớn hoặc dữ liệu cần ghi không có sẵn. Chẳng hạn, việc ghi nhật ký hệ thống (*log*) chỉ được thực hiện khi có lỗi hoặc ngoại lệ xảy ra trong thời gian chương trình thực thi.

Để giải quyết tình huống này, .NET Framework giới thiệu khái niệm luồng dữ liệu và các lớp StreamReader, StreamWriter để đọc và ghi dữ liệu từ luồng một cách hiệu quả. Trong ngữ cảnh thao tác với các thiết bị nhập/xuất, luồng (*stream*) được dùng để biểu diễn một dòng dữ liệu từ một nguồn (chương trình) đến một đích nào đó (tập tin). Các luồng cung cấp một cách thức thống nhất để tương tác với một chuỗi các byte.

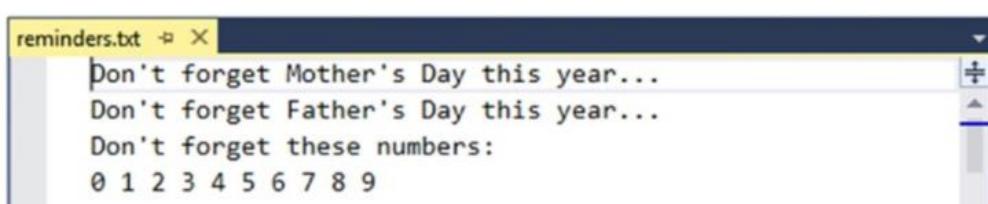
Hai lớp StreamReader và StreamWriter rất tiện dụng trong việc đọc và ghi dữ liệu từ một luồng ký tự. Mặc định, cả hai lớp đều làm việc với các ký tự Unicode. Tuy nhiên, điều này có thể thay đổi bằng cách thiết lập giá trị thích hợp cho thuộc tính Encoding. Lớp StreamReader cung cấp khả năng đọc và xem một ký tự từ luồng. Lớp StreamWriter cho phép ghi dữ liệu có nhiều kiểu khác nhau vào luồng. Lớp này có chứa thuộc tính AutoFlush, cho phép StreamWriter đẩy toàn bộ dữ liệu trong luồng đến đích nhận mỗi khi một thao tác ghi được thực hiện. Chương trình 5.2 tạo một tập tin tên là reminders.txt chứa 4 dòng dữ liệu. Ba dòng đầu là các chuỗi ký tự. Dòng cuối cùng là 10 số từ 0 đến 9.

Chương trình 5.2. Tạo tập tin từ chuỗi ký tự và số

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("* Example StreamWriter / StreamReader\n");
        // Get a StreamWriter and write string data.
        using (StreamWriter writer = File.CreateText("reminders.txt"))
        {
            writer.WriteLine("Don't forget Mother's Day this year...");
            writer.WriteLine("Don't forget Father's Day this year...");
            writer.WriteLine("Don't forget these numbers:");
            for (int i = 0; i < 10; i++)
                writer.Write(i + " ");
            // Insert a new line.
            writer.WriteLine();
        }
        Console.WriteLine("Created file and wrote some thoughts...");
        Console.ReadLine();
    }
}
```

Kết quả được thể hiện như trong Hình 5.1.



Hình 5.1. Kết quả đọc tập tin văn bản

Để đọc dữ liệu từ tập tin vừa tạo dùng StreamReader, ta thực hiện theo Chương trình 5.3 như sau:

Chương trình 5.3. Đọc tập tin sử dụng StreamReader

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("/**Example StreamWriter / StreamReader ***/\n");
        // Now read data from file.
        Console.WriteLine("Here are your thoughts:\n");
        using (StreamReader sr = File.OpenText("reminders.txt"))
        {
            string input = null;
            while ((input = sr.ReadLine()) != null)
            {
                Console.WriteLine(input);
            }
        }

        Console.ReadLine();
    }
}
```

5.2. Đọc và ghi tập tin định dạng nhị phân

.NET Framework cung cấp hai lớp BinaryReader và BinaryWriter nhằm giúp đơn giản hóa việc đọc và ghi nội dung vào một tập tin dưới dạng nhị phân. Cả hai lớp này đều dẫn xuất trực tiếp từ lớp System.Object. Lớp BinaryWriter định nghĩa phương thức Write với nhiều biến thể để cho phép ghi một giá trị dữ liệu vào luồng bên dưới (ở đây là tập tin). Trong khi đó, lớp BinaryReader cung cấp phương thức Read() và ReadXxx() để cho phép đọc dữ liệu từ luồng và chuyển đổi thành kiểu dữ liệu .Net. Cả hai lớp này đều hỗ trợ khả năng truy xuất ngẫu nhiên. Bảng 5.2 và bảng 5.3 liệt kê các phương thức quan trọng của hai lớp vừa đề cập.

Bảng 5.2. Các thành phần quan trọng của lớp BinaryWriter

Thuộc tính/ Phương thức	Ý nghĩa
BaseStream	Thuộc tính chỉ đọc. Cung cấp khả năng truy xuất tới luồng kết nối tập tin với đối tượng BinaryWriter.
Close()	Đóng luồng
Flush()	Đẩy toàn bộ nội dung đã ghi xuống luồng hiện tại
Seek()	Thiết lập vị trí con trỏ ghi trong luồng hiện tại
Write()	Ghi giá trị dữ liệu vào luồng hiện tại

Bảng 5.3. Các thành phần quan trọng của lớp BinaryReader

Thuộc tính/ Phương thức	Ý nghĩa
BaseStream	Thuộc tính chỉ đọc. Cung cấp khả năng truy xuất tới luồng kết nối tập tin với đối tượng BinaryReader.
Close()	Đóng luồng
PeekChar()	Trả về ký tự tiếp theo nhưng không di chuyển con trỏ đọc đến vị trí kế tiếp
Read()	Đọc một số lượng byte hoặc ký tự và lưu vào một mảng
Write()	Đọc một giá trị từ luồng và chuyển đổi sang kiểu dữ liệu phù hợp. Ví dụ: ReadByte(), ReadBoolean(), ReadInt32(), ...

Chương trình 5.4 cho thấy cách ghi ba giá trị số thực, số nguyên và chuỗi vào tập tin nhị phân tên là BinFile.dat.

Chương trình 5.4. Tạo tập tin nhị phân dùng BinaryWriter

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("* Example Binary Writers / Readers **\n");

        // Open a binary writer for a file.
        FileInfo f = new FileInfo("BinFile.dat");

        using (BinaryWriter bw = new BinaryWriter(f.OpenWrite()))
        {
            // Print out the type of BaseStream.
            // (System.IO.FileStream in this case).
            Console.WriteLine("Base stream is: {0}", bw.BaseStream);

            // Create some data to save in the file.
            double aDouble = 1234.67;
            int anInt = 34567;
            string aString = "A, B, C";

            // Write the data.
            bw.Write(aDouble);
            bw.Write(anInt);
            bw.Write(aString);
        }
        Console.WriteLine("Done!");
        Console.ReadLine();
    }
}
```

Để đọc dữ liệu đã ghi từ tập tin BinFile.dat, ta sử dụng lớp BinaryReader được minh họa trong Chương trình 5.5 như sau:

Chương trình 5.5. Đọc dữ liệu từ tập tin nhị phân sử dụng BinaryReader

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        FileInfo f = new FileInfo("BinFile.dat");

        // Read the binary data from the stream.
        using (BinaryReader br = new BinaryReader(f.OpenRead()))
        {
            Console.WriteLine(br.ReadDouble());
            Console.WriteLine(br.ReadInt32());
            Console.WriteLine(br.ReadString());
        }

        Console.ReadLine();
    }
}
```

5.3. Đọc và ghi nội dung từ tập tin XML

XML là một ngôn ngữ có khả năng tự mô tả. Tập tin XML vừa chứa dữ liệu, vừa chứa các quy tắc và thông tin để có thể rút trích được dữ liệu từ tập tin đó. Có hai cách phổ biến để thao tác với các tập tin XML. Thứ nhất là dùng các lớp được cung cấp sẵn bởi .NET Framework trong namespace System.Xml. Bạn phải nạp namespace này vào mã nguồn trước khi sử dụng các lớp để đọc, ghi nội dung XML. Thứ hai là sử dụng LINQ to XML. LINQ to XML là một tập hợp các thư viện hàm cho phép bạn nạp tập tin XML vào bộ nhớ để xử lý và cập nhật tài liệu XML một cách thuận tiện và hiệu quả. Để sử dụng LINQ to XML, bạn cần nạp namespace System.Xml.Linq vào mã nguồn. Ngoài ra, còn có một số thư viện của bên thứ 3 nhằm giúp xử lý nội dung XML đơn giản và dễ dàng hơn.

Việc đọc tập tin XML và phân tích nội dung bên trong các thẻ XML có thể được thực hiện bằng nhiều cách khác nhau tùy thuộc vào yêu cầu cụ thể. Bạn có thể sử dụng lớp XmlDocument, XmlReader, XmlTextReader, XmlDocument trong namespace System.Xml. Hoặc bạn có thể sử dụng các lớp XDocument, XElement trong namespace System.Xml.Linq. Phần này giới thiệu cách sử dụng lớp XmlDocument để đọc, phân tích nội dung và truy vấn tài liệu XML sử dụng XPath. Lớp XmlDocument đọc toàn bộ nội dung XML vào bộ nhớ và cho phép chúng ta đọc nội dung các thẻ bất kỳ, điều hướng qua lại giữa phần tử cha-con, các phần tử cùng cấp.

Giả sử ta có tập tin books.xml như dưới đây:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Books.xml stores information about Mahesh Chand and related books --&gt;
&lt;catalog&gt;
    &lt;book ISBN="9831123212" yearpublished="2002"&gt;
        &lt;title&gt;A Programmer's Guide to ADO .Net using C#&lt;/title&gt;
        &lt;author&gt;
            &lt;first-name&gt;Mahesh&lt;/first-name&gt;
            &lt;last-name&gt;Chand&lt;/last-name&gt;
        &lt;/author&gt;
        &lt;publisher&gt;Apress&lt;/publisher&gt;
        &lt;price&gt;44.99&lt;/price&gt;
    &lt;/book&gt;
    &lt;book ISBN="9781484234" yearpublished="2019"&gt;
        &lt;title&gt;Pro Entity Framework Core 2&lt;/title&gt;
        &lt;author&gt;
            &lt;first-name&gt;Adam&lt;/first-name&gt;
            &lt;last-name&gt;Freeman&lt;/last-name&gt;
        &lt;/author&gt;
        &lt;publisher&gt;Apress&lt;/publisher&gt;
        &lt;price&gt;45.09&lt;/price&gt;
    &lt;/book&gt;
&lt;/catalog&gt;</pre>
```

Để đọc nội dung XML và lấy thông tin danh mục sách trong tập tin nói trên, ta có thể sử dụng lớp XmlDocument như Chương trình 5.6:

Chương trình 5.6. Đọc dữ liệu từ tập tin XML sử dụng XmlDocument

```
public static void Main(string[] args)
{
    // Load XML file into XmlDocument instance
    var xmlDoc = new XmlDocument();
    xmlDoc.Load("../..\\books.xml");

    // Get list of nodes whose name is Book
    var nodeList = xmlDoc.DocumentElement.SelectNodes("/catalog/book");

    foreach (XmlNode node in nodeList)
    {
        // Read attribute value
        var isbn = node.Attributes["ISBN"].Value;
        // Read child node value
        var title = node.SelectSingleNode("title").InnerText;
        var price = node.SelectSingleNode("price").InnerText;
        // Read the descendant node value
        var firstName = node.SelectSingleNode("author/first-
name").InnerText;
        var lastName = node.SelectSingleNode("author/last-
name").InnerText;
        Console.WriteLine("{0,-15}{1,-50}{2,-15}{3,-15}{4,6}",
                        isbn, title, firstName, lastName, price);
```

```
    }  
}
```

Kết quả đọc như sau:

9831123212	A Programmer's Guide to ADO .Net using C#	Mahesh	Chand	44.99
9781484234	Pro Entity Framework Core 2	Adam	Freeman	45.09

Tương tự, việc tạo tập tin XML từ dữ liệu cho trước cũng tương đối đơn giản. Bạn có thể sử dụng các lớp trong namespace System.Xml hoặc sử dụng LINQ to XML tùy theo yêu cầu dự án. Chương trình 5.7 minh họa cách sử dụng lớp XmlWriter để ghi dữ liệu ra một tập tin books.xml.

Chương trình 5.7. Sử dụng lớp XmlWriter để ghi dữ liệu ra một tập tin

```
public static void Main(string[] args)  
{  
    using (XmlWriter writer = XmlWriter.Create("books.xml"))  
    {  
        // Write Processing Instruction  
        String pi = "type=\"text/xsl\" href=\"book.xsl\"";  
        writer.WriteProcessingInstruction("xmlstylesheet", pi);  
        // Write DocumentType  
        writer.WriteDocType("catalog", null, null, "<!ENTITY h  
\"hardcover\">");  
        // Write a Comment  
        writer.WriteComment("This is a book sample XML");  
        // Root element - start tag  
        writer.WriteStartElement("book");  
        // Write ISBN attribute  
        writer.WriteString("ISBN", "9831123212");  
        // Write year attribute  
        writer.WriteString("yearpublished", "2002");  
        // Write title  
        writer.WriteString("author", "Mahesh Chand");  
        // Write author  
        writer.WriteString("title", "Visual C# Programming");  
        // Write price  
        writer.WriteString("price", "44.95");  
        // Root element - end tag  
        writerEndElement();  
        // End Document  
        writer.EndDocument();  
        // Flush it  
        writer.Flush();  
    }  
}
```

Tập tin books.xml được tạo bởi đoạn chương trình trên có nội dung như sau:

```
<?xml version="1.0" encoding="utf-8"?>  
<?xml-stylesheet type="text/xsl" href="book.xsl"?>
```

```
<!DOCTYPE book [
    <!ENTITY h "hardcover">
]>
<!--This is a book sample XML-->
<book ISBN="9831123212" yearpublished="2002">
    <author>Mahesh Chand</author>
    <title>Visual C# Programming</title>
    <price>44.95</price>
</book>
```

Việc ghi nội dung XML như trên sẽ trở nên phức tạp và khó kiểm soát khi lượng dữ liệu lớn và có nhiều thông tin hơn. Nếu bạn có một danh sách các đối tượng và muốn lưu trữ dữ liệu của chúng dưới dạng tập tin XML, lớp XmlSerializer sẽ là một lựa chọn phù hợp. Giả sử, ta có lớp Book chứa 5 thuộc tính mô tả về một cuốn sách như sau:

```
public class Book
{
    public string ISBN { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
    public decimal Price { get; set; }
    public int YearPublished { get; set; }
}
```

Khi đó, để lưu dữ liệu từ một mảng đối tượng books vào tập tin books.xml, ta sử dụng đoạn mã sau:

```
private static void SaveToXmlFile(List<Book> books)
{
    var serializer = new XmlSerializer(typeof(List<Book>));

    using (var writer = new StreamWriter("books.xml"))
    {
        serializer.Serialize(writer, books, null);
        writer.Close();
    }
}
```

Tập tin books.xml kết quả:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfBook
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Book>
        <ISBN>9831123212</ISBN>
        <Title>A Programmer's Guide to ADO .Net using C#</Title>
        <Author>Mahesh Chand</Author>
        <Price>44.99</Price>
        <YearPublished>2002</YearPublished>
    </Book>
    <Book>
```

```
<ISBN>9781484234</ISBN>
<Title>Pro Entity Framework Core 1</Title>
<Author>Adam Freeman</Author>
<Price>44.99</Price>
<YearPublished>2018</YearPublished>
</Book>
</ArrayOfBook>
```

5.4. Đọc và tạo tập tin CSV

CSV (*Comma Separated Values*) là một loại định dạng tập tin văn bản đơn giản mà trong đó, các giá trị được ngăn cách với nhau bằng dấu phẩy. Định dạng CSV thường được sử dụng để lưu các bảng tính quy mô nhỏ như danh bạ, danh sách lớp, báo cáo, ... Nó chỉ chứa những giá trị đơn thuần, bao gồm số, chữ và các ký tự khác. CSV được xem như là một tập tin phẳng (*flat file*) và gần như là một tập văn bản thông thường. Vì vậy, bạn có thể sử dụng cách thức đã học trong phần 1 để đọc và phân tích nội dung CSV.

Tuy nhiên, việc tự mình viết mã để phân tích nội dung tập tin CSV có thể trở nên rườm rà và phức tạp nếu dữ liệu trong các trường có chứa ký tự đặc biệt hoặc dấu phẩy (trùng với dấu phân tách các trường). Tin vui là cộng đồng người dùng .NET đã tạo ra nhiều thư viện hỗ trợ để giúp việc xử lý tập tin CSV trở nên vô cùng đơn giản. Một trong số đó là gói thư viện CsvHelper. Đây là một thư viện mã nguồn mở, có khả năng đọc-ghi nội dung CSV cực kỳ linh hoạt, nhanh chóng và rất dễ sử dụng. Để sử dụng CsvHelper, trước tiên, ta phải cài đặt gói thông qua Package Manager Console hoặc cửa sổ NuGet Package Manager. Để cài đặt CsvHelper từ cửa sổ dòng lệnh Package Manager Console, ta chỉ cần gõ lệnh:

```
PM> Install-Package CsvHelper
```

Để cài đặt từ cửa sổ NuGet Package Manager, ta chọn menu Projects (hoặc nhấp phải chuột vào tên dự án) rồi chọn menu Manage NuGet Packages. Sau đó, nhập tên CsvHelper vào ô tìm kiếm. Cuối cùng, chọn phiên bản mong muốn và nhấn nút Install để cài đặt gói thư viện vào dự án (Hình 5.2).

Giả sử, ta có tập tin products.csv với nội dung như sau:

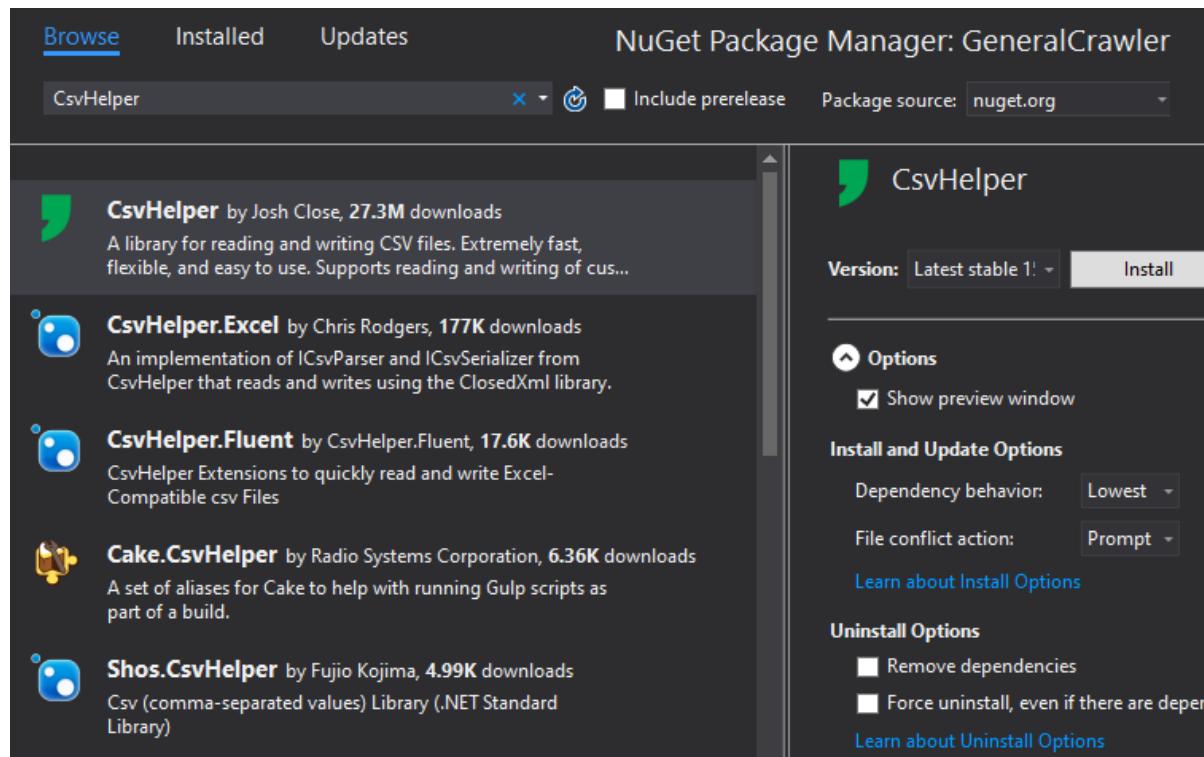
```
Id,Name
1,one
2,two
```

Khi đó, với sự hỗ trợ của thư viện CsvHelper, ta có thể dễ dàng đọc nội dung tập tin và biến đổi nó thành một danh sách các đối tượng có kiểu Product chỉ với vài dòng lệnh C# như dưới đây

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Giáo trình Lập trình Cơ sở dữ liệu

```
private static void ReadCsvFile()
{
    using (var reader = new StreamReader("products.csv"));
    using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))
    {
        var records = csv.GetRecords<Product>();
    }
}
```



Hình 5.2. Cài đặt CsvHelper từ cửa sổ NuGet Package Manager

Tương tự, việc ghi dữ liệu từ một danh sách các đối tượng vào tập tin CSV cũng có thể hoàn thành chỉ với vài câu lệnh đơn giản.

```
private static void SaveToCsvFile(List<Product> records)
{
    using (var writer = new StreamWriter("products.csv"))
    using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture))
    {
        csv.WriteRecords(records);
    }
}
```

CsvHelper hỗ trợ các thiết lập tùy chỉnh giúp việc đọc tập tin CSV trở nên linh hoạt và dễ sử dụng. Nó cho phép lập trình viên cấu hình ký tự phân cách giữa các trường cũng như ánh xạ tên và thứ tự các trường trong tập tin CSV với các thuộc tính của đối tượng đích mà dữ liệu sẽ được chuyển đổi thành. Bạn có thể đọc toàn bộ nội dung thành một tập đối tượng hoặc có thể đọc từng mẫu tin (từng dòng) hoặc từng trường cụ thể.

5.5. Đọc và tạo bảng tính Excel

Trong quá trình làm việc, các lập trình viên thường xuyên bắt gặp vấn đề nạp dữ liệu từ các bảng tính Excel vào cơ sở dữ liệu hoặc ngược lại, nhiều ứng dụng yêu cầu tính năng kết xuất dữ liệu đã xử lý ra các bảng tính Excel. Các tập tin dạng bảng tính như Excel là các tập tin có cấu trúc và tuân thủ theo định dạng chuẩn (chẳng hạn như Open Office XML). Chúng ta có thể sử dụng các thư viện hàm cung cấp sẵn trong .NET để thao tác với chúng. Tuy nhiên, việc này đòi hỏi phải có hiểu biết về định dạng OOXML và mã nguồn xử lý tập tin có thể rất dài dòng, rườm rà và có khả năng gây ra lỗi cao.

Do việc xử lý bảng tính là một tác vụ thường gặp, cộng đồng .NET đã xây dựng nhiều gói thư viện hàm nhằm đơn giản hóa các thao tác xử lý, giúp lập trình viên dễ dàng làm việc với các tập tin Excel hay Spreadsheet. Các thư viện phổ biến bao gồm: `ExcelDataReader`, `Microsoft.Office.Interop.Excel`, `EPPlus`, `LinqToExcel`, `NPOI`, ... Phần này sẽ giới thiệu sơ lược cách sử dụng gói EPPlus để đọc, ghi và định dạng dữ liệu từ một tập tin Excel (`.xlsx`).

EPPlus là một thư viện hàm chuyên dụng hỗ trợ cho việc xử lý các bảng tính có định dạng Open Office XML. Nó có thể đọc, ghi dữ liệu từ các tập tin `*.xlsx` mà không cần phải cài đặt MS Office Excel. EPPlus là một thư viện độc lập, không phụ thuộc vào bất kỳ thư viện nào khác và được phân phối thông qua Nuget.

Cho tập tin Excel có tên `products.xlsx` như sau (Hình 5.3):

	A	B	C	D
1	ID	Name	Price	Stock Date
2		1 Refrigerator	1000	5/11/2019
3		2 Television	450	11/23/2019
4		3 Air Conditioner	880	2/14/2020

Hình 5.3. Minh họa nội dung tập tin Excel

Với tập tin đã cho, đoạn mã dưới đây cho thấy EPPlus dễ dàng đọc nội dung và chuyển đổi thành một mảng các đối tượng Product.

```
private static void ReadExcel()
{
    // Create list of products
    var products = new List<Product>();

    var productFile = new FileInfo("products.xlsx");
    using (ExcelPackage excelPackage = new ExcelPackage(productFile))
    {
        //Get WorkSheet by index (EPPlus indexes are base 1, not base 0!)
        ExcelWorksheet sheet = excelPackage.Workbook.Worksheets[1];
        //Get a WorkSheet by name. Throw an exception if not existed
        var namedWorksheet =
excelPackage.Workbook.Worksheets["SheetName"];
```

```
//If you don't know if a worksheet exists, you could use LINQ, So it
//doesn't throw an exception; return null in case it doesn't find it
ExcelWorksheet anotherWorksheet =
excelPackage.Workbook.Worksheets.FirstOrDefault(x => x.Name ==
"SheetName");

// Get position of last row containing data
var endRow = sheet.Dimension.End.Row;

for (var rowIdx = 2; rowIdx < endRow; rowIdx++)
{
    // Create new product
    var item = new Product();

    // Get content from cells
    item.Id = sheet.Cells[rowIdx, 1].Value.ToString();
    item.Name = sheet.Cells[rowIdx, 2].Value.ToString();
    item.Price = decimal.Parse(sheet.Cells[rowIdx,
3].Value.ToString());
    item.StockDate = DateTime.Parse(
        sheet.Cells[rowIdx, 4].Value.ToString());

    products.Add(item);
}
}

}
```

Việc tạo một bảng tính mới và ghi dữ liệu vào bảng tính đó cũng khá đơn giản với EPPlus. Thay vì đọc nội dung từ thuộc tính Value của một ô (cell), ta gán giá trị cho nó và cuối cùng lưu nội dung vào tập tin bảng cách dùng lệnh excelPackage.Save().

```
private static void SaveToExcel()
{
    //Create a new ExcelPackage
    using (ExcelPackage excelPackage = new ExcelPackage())
    {
        //Set some properties of the Excel document
        excelPackage.Workbook.Properties.Author = "Nga Phan";
        excelPackage.Workbook.Properties.Title = "Sample Spreadsheet";
        excelPackage.Workbook.Properties.Subject = "Demo export data";
        excelPackage.Workbook.Properties.Created = DateTime.Now;

        //Create the WorkSheet
        var worksheet = excelPackage.Workbook.Worksheets.Add("Sheet 1");

        //Add some text to cell A1
        worksheet.Cells["A1"].Value = "My first EPPlus spreadsheet!";

        //You could also use [line, column] notation:
        worksheet.Cells[1, 2].Value = "This is cell B1!";

        // Write non-string value
        worksheet.Cells[1, 3].Value = 12.55m;
```

```
worksheet.Cells[1, 4].Value = DateTime.Now;

//Save your file
var fi = new FileInfo(@"data.xlsx");
excelPackage.SaveAs(fi);
}

}
```

Việc xuất dữ liệu từ một mảng các đối tượng ra tập tin Excel thậm chí còn đơn giản hơn nếu bạn muốn kết xuất giá trị trên tất cả thuộc tính của đối tượng và không quan trọng thứ tự của các trường dữ liệu.

```
private static void SaveToExcel(List<Product> products)
{
    //Create a new ExcelPackage
    using (ExcelPackage excelPackage = new ExcelPackage())
    {
        //Set some properties of the Excel document
        excelPackage.Workbook.Properties.Author = "Nga Phan";
        excelPackage.Workbook.Properties.Title = "New Products";
        excelPackage.Workbook.Properties.Subject = "New Products 2020";
        excelPackage.Workbook.Properties.Created = DateTime.Now;

        //Create the WorkSheet
        var worksheet = excelPackage.Workbook.Worksheets.Add("Sheet 1");

        //add the content from the List<Product>, starting at cell A1
        worksheet.Cells["A1"].LoadFromCollection(products);

        //Save your file
        var fi = new FileInfo(@"data.xlsx");
        excelPackage.SaveAs(fi);
    }
}
```

Đặc biệt, EPPlus cho phép lập trình viên thiết lập độ rộng của các cột, định dạng về màu sắc, font chữ cho các ô trong bảng tính. Nó cũng hỗ trợ việc sử dụng các công thức của Excel và tính giá trị cho các ô có sử dụng công thức.

```
//Sets the numberformat for a range
worksheet.Cells["A1:B3"].Style.NumberFormat.Format = "#,##0";

//Same as above,A1:B3
worksheet.Cells[1, 1, 3, 2].Style.NumberFormat.Format = "#,##0";

//Sets the numberformat for a range containing two addresses.
worksheet.Cells["A1:B3,D1:E57"].Style.NumberFormat.Format = "#,##0";

//Sets font-bold to true for column A & B
worksheet.Cells["A:B"].Style.Font.Bold = true;

//Sets font-bold to true for row 1,column A and cell C3
```

```
worksheet.Cells["1:1,A:A,C3"].Style.Font.Bold = true;  
  
//Sets font to Arial for all cells in a worksheet.  
worksheet.Cells["A:XFD"].Style.Font.Name = "Arial";  
  
//This is equal to the above.  
worksheet.Cells.Style.Font.Name = "Arial";  
  
//Address "A1:A5"  
using (var range = worksheet.Cells[1, 1, 1, 5])  
{  
    // Merge cells from A1 to A5  
    range.Merge = true;  
  
    // Set font bold  
    range.Style.Font.Bold = true;  
  
    // Set background color and text color  
    range.Style.Fill.PatternType = ExcelFillStyle.Solid;  
    range.Style.Fill.BackgroundColor.SetColor(Color.DarkBlue);  
    range.Style.Font.Color.SetColor(Color.White);  
}  
  
// Formula: Set the total value of cells in range A1 - A25 into A27  
worksheet.Cells["A1"].Value = 1;  
worksheet.Cells["A2"].Value = 2;  
worksheet.Cells["A3"].Formula = "=SUM(A1:A2)";  
  
// calculates all formulas in the entire workbook  
excelPackage.Workbook.Calculate();  
var result = worksheet.Cells["A3"].Value; // result = 3
```

Để biết chi tiết và đầy đủ tính năng của thư viện này, bạn có thể tham khảo thêm tại website chính thức <https://www.epplussoftware.com/> và các website khác cung cấp các hướng dẫn chi tiết về cách sử dụng thư viện, chẳng hạn: <https://riptutorial.com/epplus>.

5.6. Entity Framework 6

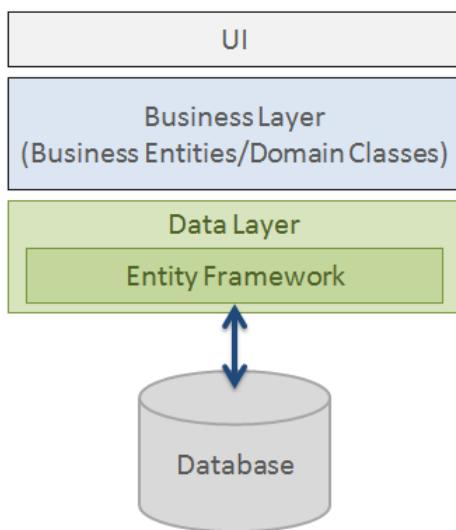
Trước .NET 3.5, các lập trình viên thường viết mã ADO .Net hoặc Enterprise Data Access Block để truy vấn hoặc lưu trữ dữ liệu từ ứng dụng vào cơ sở dữ liệu. Chúng ta đã quen với việc tạo một kết nối tới cơ sở dữ liệu, tạo một DataSet để lưu trữ các mẫu tin được lấy về từ cơ sở dữ liệu hoặc để truyền dữ liệu vào cơ sở dữ liệu, chuyển đổi dữ liệu từ DataSet thành các đối tượng trong .NET hoặc ngược lại để áp dụng các quy tắc nghiệp vụ (business rules). Đây là một quá trình khá rườm rà và dễ gây ra lỗi. Vì vậy, Microsoft đã cung cấp một thư viện mới gọi là Entity Framework để tự động hóa mọi thao tác liên quan đến cơ sở dữ liệu trong ứng dụng của bạn. Tài liệu chính thức về Entity Framework bạn có thể tham khảo từ các website chính thức là <https://www.entityframeworktutorial.net/> và <https://docs.microsoft.com/en-us/ef/ef6/>. Nội dung, hình ảnh và các đoạn mã dùng để minh họa trong phần này được tham khảo, dịch lại và trích dẫn từ website <https://www.entityframeworktutorial.net/>

Entity Framework (gọi tắt là EF) là một thư viện ORM (Object-Relational Mapper) mã nguồn mở dành cho các ứng dụng .NET và được hỗ trợ bởi Microsoft. Nó cho phép các lập trình viên làm việc với dữ liệu thông qua các đối tượng của các lớp trong một miền ứng dụng cụ thể mà không cần phải quá chú tâm vào các bảng và các cột bên dưới cơ sở dữ liệu và cách mà dữ liệu được lưu trữ. Với Entity Framework, các lập trình viên có thể làm việc ở mức trừu tượng hóa cao hơn khi họ thao tác với dữ liệu, có thể tạo ra và bảo trì các ứng dụng có sử dụng cơ sở dữ liệu mà không cần phải viết nhiều mã nguồn như các ứng dụng truyền thống.

5.6.1. Định nghĩa

Entity Framework là một công cụ ánh xạ đối tượng-quan hệ (ORM – Object-Relational Mapper) cho phép các lập trình viên .NET làm việc với cơ sở dữ liệu thông qua các đối tượng trong .NET. Nó giúp loại bỏ hầu hết các đoạn mã truy cập dữ liệu mà các lập trình viên thường phải viết để truy vấn dữ liệu.

Hình 5.4 minh họa cách áp dụng Entity Framework vào ứng dụng của chúng ta. Trong mô hình này, Entity Framework nằm ở tầng truy xuất dữ liệu (*Data Layer*), làm nhiệm vụ trung chuyển dữ liệu giữa thành phần nghiệp vụ (*Business Layer*) và cơ sở dữ liệu (*Database*). Tầng nghiệp vụ chứa các thực thể nghiệp vụ hay các lớp biểu diễn các thực thể trong miền ứng dụng. Entity Framework lưu các giá trị được chứa trong các thuộc tính của các thực thể nghiệp vụ vào các cột của các bảng trong cơ sở dữ liệu hoặc ngược lại, lấy các mẫu tin từ cơ sở dữ liệu và biến đổi nó thành các đối tượng thực thể nghiệp vụ một cách hoàn toàn tự động.



Hình 5.4. Kiến trúc cơ bản của ứng dụng

5.6.2. Các tính năng của Entity Framework

- *Đa nền tảng*: EF Core là một thư viện đa nền tảng (cross-platform framework), có thể chạy được trên Windows, Linux và Mac;
- *Khả năng mô hình hóa*: EF có thể tạo ra một mô hình dữ liệu thực thể (EDM – Entity Data Model) dựa trên các đối tượng CLR đơn giản (POCO – Plain

Old CLR Object) với các thuộc tính get/set có kiểu dữ liệu khác nhau. EF sử dụng mô hình này để truy vấn và lưu trữ dữ liệu của các thực thể vào cơ sở dữ liệu;

- *Truy vấn:* EF cho phép chúng ta sử dụng các truy vấn LINQ (Language Integrated Query) để lấy dữ liệu từ cơ sở dữ liệu. Trình xử lý cơ sở dữ liệu (database provider) sẽ dịch các truy vấn LINQ thành các câu truy vấn phù hợp với từng loại cơ sở dữ liệu cụ thể (chẳng hạn như SQL đối với cơ sở dữ liệu quan hệ). EF cũng cho phép thực thi trực tiếp một truy vấn SQL đến cơ sở dữ liệu;
- *Theo dõi các thay đổi:* EF theo dõi các thay đổi xảy ra trong các đối tượng (cụ thể là giá trị các thuộc tính) để biết những gì cần phải được lưu trữ ngược trở lại vào cơ sở dữ liệu;
- *Lưu trữ:* EF biết cách thực thi các câu lệnh INSERT, UPDATE, DELETE đến cơ sở dữ liệu dựa trên các thay đổi xảy ra trên các thực thể khi phương thức SaveChanges() được gọi. Ngoài ra, EF cũng hỗ trợ phương thức bất đồng bộ SaveChangesAsync();
- *Kiểm soát truy vấn đồng thời:* Theo mặc định, EF sử dụng phương pháp kiểm soát truy vấn đồng thời tối ưu (Optimistic Concurrency) để bảo vệ các thay đổi bị ghi đè bởi một người dùng khác kể từ khi dữ liệu được lấy từ cơ sở dữ liệu và nạp lên ứng dụng;
- *Giao tác/giao dịch:* EF thực hiện việc quản lý giao dịch cơ sở dữ liệu một cách tự động trong quá trình truy vấn hay lưu trữ dữ liệu. Nó cũng cung cấp các tùy chọn và phương tiện để có thể tùy biến việc quản lý các giao dịch.
- *Caching:* EF cũng cung cấp lưu trữ dữ liệu vào bộ đệm. Vì vậy, các truy vấn lặp lại sẽ trả về kết quả lấy từ bộ đệm thay vì phải đọc lại từ cơ sở dữ liệu.
- *Có bộ quy ước xây dựng sẵn:* EF tuân theo các quy ước về mẫu lập trình theo cấu hình (configuration programming pattern) và bao gồm một bộ các quy tắc để tự động hóa việc cấu hình mô hình cho EF.
- *Cấu hình:* EF cho phép chúng ta cấu hình các mô hình EF bằng cách dùng các thuộc tính chú thích cho dữ liệu (data annotation attribute) hoặc sử dụng một tập các giao diện lập trình ứng dụng (Fluent API) linh hoạt để tùy biến, ghi đè các quy ước mặc định.
- *Tích hợp và di chuyển:* EF cung cấp sẵn một tập các lệnh tích hợp – có thể được thực thi trong NuGet Package Manager Console hoặc Command Line Interface – để tạo ra và quản lý các lược đồ cơ sở dữ liệu.

5.6.3. Các phiên bản

Microsoft giới thiệu Entity Framework lần đầu tiên vào năm 2008, đi kèm với .NET Framework 3.5 và sau đó nhiều phiên bản kế tiếp được cho ra đời. Hiện nay, hai phiên bản cuối của Entity Framework là EF 6 và EF Core. Bảng 5.4. liệt kê một số khác biệt quan trọng giữa hai nhánh này của EF.

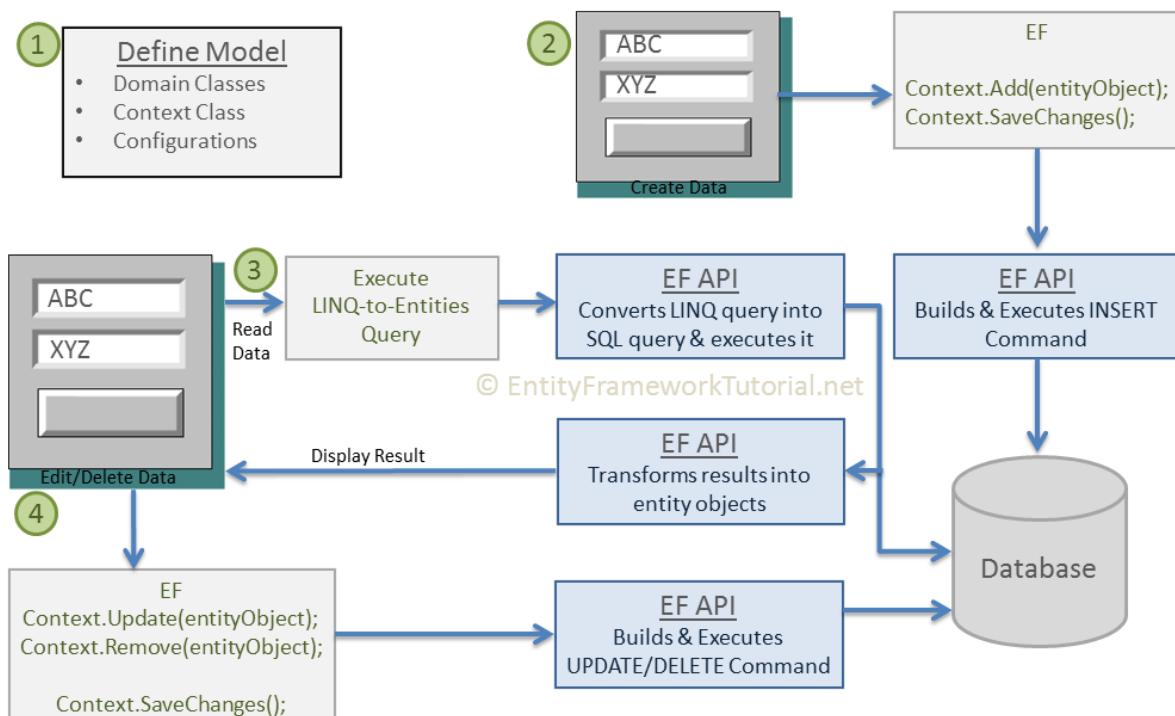
Bảng 5.4. Sự khác biệt giữa EF6 và EF Core

EF6	EF Core
Phát hành lần đầu tiên năm 2008 với .NET Framework 3.5 SP1	Phát hành lần đầu tiên năm 2008 với .NET Framework 3.5 SP1
Ôn định và nhiều tính năng	Mới và cải tiến
Chỉ sử dụng cho Windows	Windows, Linux, OSX
Làm việc với .NET Framework 3.5 trở lên	Làm việc với .NET Framework 4.5 và .NET Core

5.6.4. Luồng xử lý trong Entity Framework

Hình 5.5 cho thấy hai luồng xử lý căn bản xảy ra trong Entity Framework:

- Đầu tiên, lập trình viên cần định nghĩa một mô hình. Việc định nghĩa mô hình bao gồm việc tạo ra các lớp cho miền ứng dụng (*domain classes*), lớp Context kế thừa từ lớp DbContext và các cấu hình nếu có. EF sẽ thực hiện các thao tác truy vấn, thêm, xóa hay cập nhật dữ liệu (*CRUD operations*) dựa trên mô hình đó.
- Để thêm dữ liệu, thực thể nghiệp vụ (hay đối tượng trong miền ứng dụng) được thêm vào đối tượng Context và gọi hàm SaveChanges(). Giao diện lập trình ứng dụng của EF sẽ xây dựng một câu lệnh INSERT thích hợp và thực thi nó để chèn dữ liệu vào cơ sở dữ liệu.



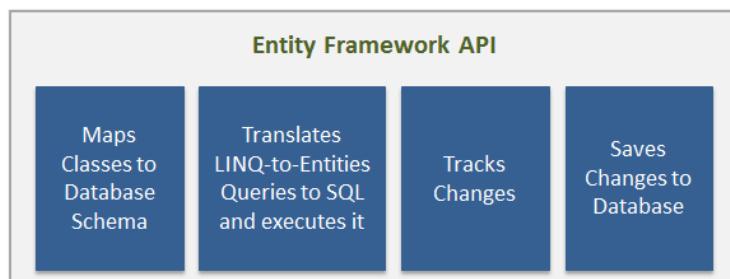
Hình 5.5. Luồng xử lý trong Entity Framework

- Để đọc dữ liệu, lập trình viên cần viết và thực thi một truy vấn LINQ-to-Entities. EF sẽ chuyển đổi truy vấn này thành câu lệnh truy vấn SQL đối với cơ

sở dữ liệu quan hệ và thực thi nó. Kết quả sẽ được biến đổi thành các đối tượng trong miền ứng dụng và hiển thị lên giao diện người dùng;

- Để cập nhật hay xóa dữ liệu, ta gọi phương thức tương ứng để cập nhật hoặc xóa các thực thể khỏi đối tượng Context và gọi phương thức SaveChanges(). EF sẽ xây dựng các câu lệnh UPDATE và DELETE thích hợp để thực thi chúng trong cơ sở dữ liệu.

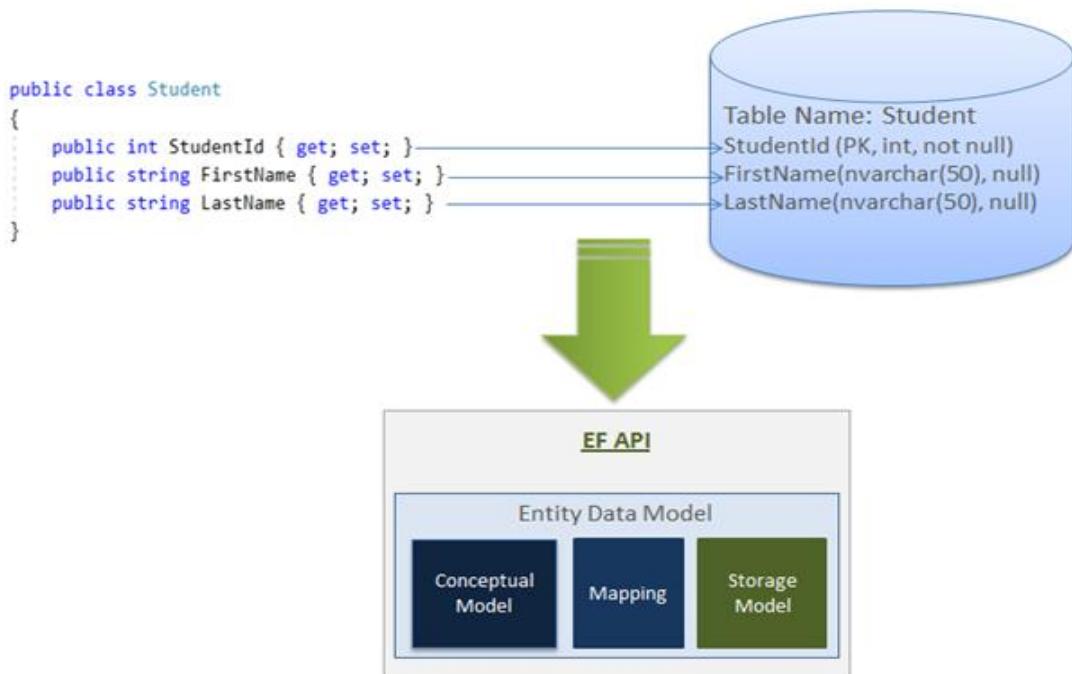
Giao diện lập trình ứng dụng của EF (gồm EF6 và EF Core) có khả năng ánh xạ các lớp trong miền ứng dụng thành lược đồ cơ sở dữ liệu, biến đổi và thực thi các lệnh truy vấn LINQ thành lệnh SQL, theo dõi các thay đổi xảy ra trên các thực thể và lưu trữ các thay đổi vào cơ sở dữ liệu (Hình 5.6).



Hình 5.6. Các chức năng của Entity Framework API

5.6.5. Mô hình dữ liệu thực thể

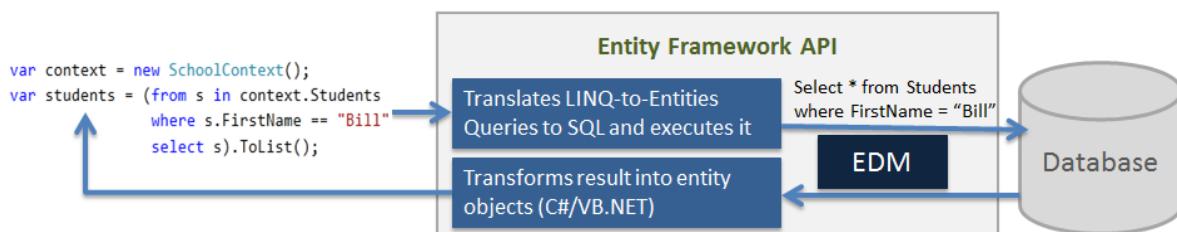
Công việc đầu tiên của EF API là xây dựng một mô hình dữ liệu thực thể (*EDM – Entity Data Model*). EDM là một cách biểu diễn trong bộ nhớ của toàn bộ siêu dữ liệu liên quan tới mô hình khái niệm, mô hình lưu trữ và ánh xạ giữa chúng (Hình 5.7).



Hình 5.7. Mô hình thực thể dữ liệu EDM

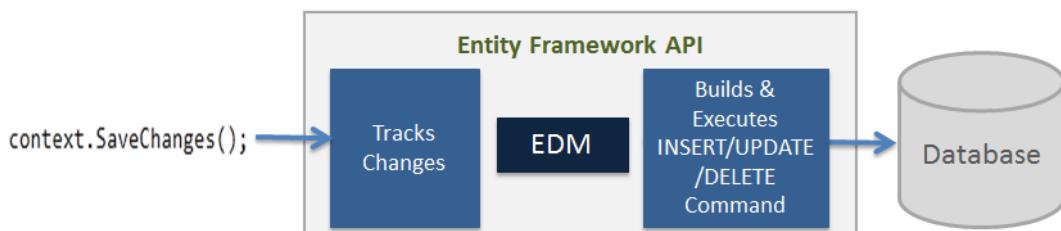
- *Mô hình khái niệm:* EF xây dựng mô hình khái niệm từ các lớp trong miền ứng dụng, lớp ngũ cành (Context), các cấu hình và các quy ước mặc định cần phải tuân thủ trong các lớp nghiệp vụ;
- *Mô hình lưu trữ:* EF xây dựng mô hình lưu trữ cho lược đồ cơ sở dữ liệu bên dưới. Theo hướng tiếp cận code-first, mô hình này được suy ra từ mô hình khái niệm. Theo tiếp cận database-first, mô hình này sẽ được tạo ra từ cơ sở dữ liệu cho trước;
- *Các ánh xạ:* EF bao gồm các thông tin ánh xạ từ mô hình khái niệm sang lược đồ cơ sở dữ liệu (mô hình lưu trữ).

EF thực hiện các thao tác CRUD sử dụng mô hình dữ liệu thực thể này bằng cách xây dựng các truy vấn SQL từ các truy vấn LINQ, sinh ra các lệnh INSERT, UPDATE, DELETE và biến đổi kết quả lấy được từ cơ sở dữ liệu thành các đối tượng trong miền ứng dụng. EF dịch các truy vấn LINQ-to-Entities thành các truy vấn SQL đối với cơ sở dữ liệu quan hệ sử dụng EDM và biến đổi kết quả ngược trở lại thành các đối tượng thực thể (Hình 5.8).



Hình 5.8. Vai trò của EDM đối với thao tác CRUD

Khi phương thức SaveChanges() được gọi, các câu lệnh INSERT, UPDATE, DELETE được tạo ra dựa vào trạng thái của các thực thể. EF có khả năng theo dõi trạng thái của từng thực thể mỗi khi một thao tác nào đó được thực hiện (Hình 5.9).



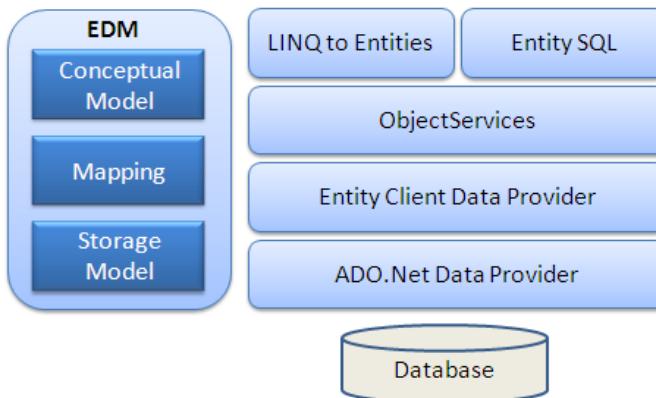
Hình 5.9. Vai trò của EDM khi gọi phương thức SaveChange()

5.6.6. Kiến trúc của Entity Framework

Hình 5.10 cho thấy kiến trúc tổng quát của Entity Framework. Kiến trúc này bao gồm các thành phần như sau:

- *EDM:* Mô hình dữ liệu thực thể gồm 3 thành phần chính: Mô hình khái niệm, mô hình lưu trữ và ánh xạ;

- *LINQ to Entities (L2E)*: Là một ngôn ngữ truy vấn được sử dụng để viết các truy vấn trên mô hình đối tượng. Nó trả về các thực thể được định nghĩa trong mô hình khái niệm;
- *Entity SQL*: Là một ngôn ngữ truy vấn khác tương tự như LINQ to Entities nhưng chỉ dùng cho EF 6. Ngôn ngữ này khó sử dụng hơn L2E;
- *Object Service*: Là điểm chính trong việc truy xuất dữ liệu từ cơ sở dữ liệu và trả chúng về cho ứng dụng. Nó có nhiệm vụ biến đổi dữ liệu được trả về từ một trình xử lý dữ liệu thành một đối tượng;
- *Entity Client Data Provider*: Nhiệm vụ chính của tầng này là chuyển đổi các truy vấn LINQ to Entities hoặc Entity SQL thành các truy vấn SQL mà cơ sở dữ liệu bên dưới có thể hiểu và thực thi được. Nó giao tiếp với trình xử lý dữ liệu ADO .Net để gửi và nhận dữ liệu từ cơ sở dữ liệu;
- *ADO .Net Data Provider*: Lớp này giao tiếp với cơ sở dữ liệu thông qua thư viện ADO .Net chuẩn.



Hình 5.10. Kiến trúc tổng quát của Entity Framework

5.6.7. Lớp ngữ cảnh trong Entity Framework

Lớp ngữ cảnh là lớp quan trọng nhất khi làm việc với EF6 và EF Core. Nó biểu diễn phiên làm việc với cơ sở dữ liệu bên dưới ứng dụng, giúp thực hiện được các thao tác thêm, cập nhật, xóa hoặc đọc dữ liệu. Lớp ngữ cảnh trong Entity Framework là một lớp kế thừa từ lớp System.Data.Entity.DbContext. Một thể hiện của lớp ngữ cảnh hay nói khác hơn là một đối tượng của lớp DbContext biểu diễn các mẫu Unit Of Work và Repository trong đó nó có thể kết hợp nhiều thay đổi và thực hiện các thay đổi đó dưới một giao dịch cơ sở dữ liệu duy nhất. Lớp ngữ cảnh được sử dụng để truy vấn hoặc lưu dữ liệu vào cơ sở dữ liệu. Nó cũng được dùng để cấu hình các lớp trong miền ứng dụng, các ánh xạ tới cơ sở dữ liệu, thiết lập việc theo dõi các thay đổi, lưu trữ tạm vào bộ đệm (caching) và thực thi giao dịch cơ sở dữ liệu, ...

Lớp SchoolContext sau đây cho thấy một ví dụ về lớp ngữ cảnh. Trong ví dụ này, lớp SchoolContext kế thừa từ lớp DbContext. Vì vậy, nó được xem là một lớp ngữ cảnh. Lớp này cũng định nghĩa 3 thuộc tính tương ứng với 3 tập thực thể: sinh viên (Student), địa chỉ của sinh viên (StudentAddress), xếp loại sinh viên (Grade).

```
using System.Data.Entity;

public class SchoolContext : DbContext // Lớp SchoolContext kế thừa từ lớp
DbContext
{
    public SchoolContext()
    {
    }
    // Các thực thể
    public DbSet<Student> Students { get; set; }
    public DbSet<StudentAddress> StudentAddresses { get; set; }
    public DbSet<Grade> Grades { get; set; }
}
```

5.6.8. Thực thể

Một thực thể trong Entity Framework là một lớp ánh xạ tới (hoặc biểu diễn) một bảng trong cơ sở dữ liệu. Lớp này phải được dùng trong một thuộc tính có kiểu DbSet< TEntity > trong lớp ngữ cảnh. Giao diện lập trình ứng dụng của EF ánh xạ mỗi thực thể đến một bảng và mỗi thuộc tính của thực thể tới một cột của bảng đó trong cơ sở dữ liệu. Chẳng hạn, xét hai lớp Student và Grade trong ứng dụng quản lý trường học như sau:

```
public class Student // Lớp Student và các thuộc tính
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[] Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public Grade Grade { get; set; }
}

public class Grade // Lớp Grade và các thuộc tính
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

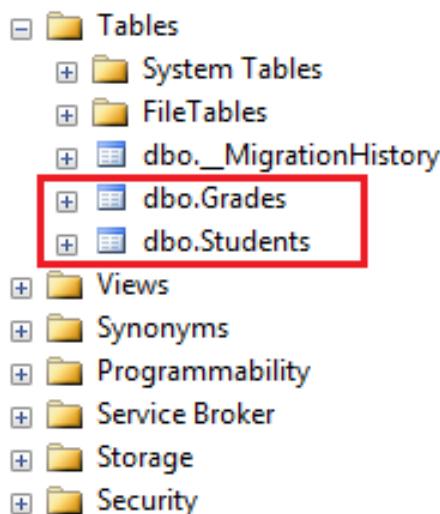
    public ICollection<Student> Students { get; set; }
}
```

Các lớp này trở thành các thực thể khi chúng được dùng trong các thuộc tính DbSet< TEntity > của lớp ngữ cảnh SchoolContext.

```
public class SchoolContext : DbContext
{
    public SchoolContext()
```

```
{  
}  
  
public DbSet<Student> Students { get; set; } //Là một thuộc tính có kiểu  
DbSet< TEntity >  
public DbSet<Grade> Grades { get; set; } //Là một thuộc tính có kiểu  
DbSet< TEntity >  
}
```

Trong lớp ngữ cảnh SchoolContext, các thuộc tính Students và Grades có kiểu DbSet< TEntity > được gọi là các tập thực thể. Student và Grade là các thực thể. EF API sẽ tạo ra các bảng Students và Grades tương ứng trong cơ sở dữ liệu (Hình 5.11).



Hình 5.11. Bảng Students và Grades được tạo trong cơ sở dữ liệu

Một thực thể có thể chứa hai loại thuộc tính: thuộc tính vô hướng (*scalar properties*) và thuộc tính điều hướng (*navigation properties*).

- *Thuộc tính vô hướng*

Thuộc tính có kiểu dữ liệu nguyên thủy (*primitive type property*) được gọi là thuộc tính vô hướng. Mỗi thuộc tính vô hướng ánh xạ đến một cột trong bảng bên dưới cơ sở dữ liệu và nó lưu một giá trị dữ liệu thực sự. Chẳng hạn, StudentId, StudentName, DateOfBirth, Photo, Height, Weight là các thuộc tính vô hướng của lớp thực thể Student.

EF API sẽ tạo ra một cột trong một bảng của cơ sở dữ liệu, tương ứng với mỗi thuộc tính vô hướng trong lớp thực thể. Chẳng hạn đoạn mã nguồn sau sẽ được ánh xạ vào bảng cơ sở dữ liệu như trong Hình 5.12.

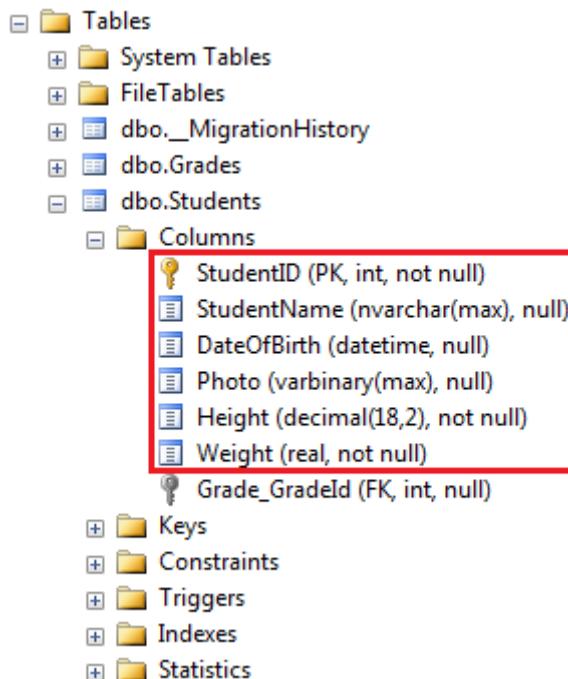
```
public class Student  
{  
    // Danh sách các thuộc tính vô hướng  
    public int StudentID { get; set; } // Mã số sinh viên  
    public string StudentName { get; set; } // Tên sinh viên  
    public DateTime? DateOfBirth { get; set; } // Ngày sinh
```

```

public byte[] Photo { get; set; } // Hình ảnh
public decimal Height { get; set; } // Chiều cao
public float Weight { get; set; } // Cân nặng

// Danh sách các thuộc tính điều hướng
public Grade Grade { get; set; }
}

```



Hình 5.12. Các thuộc tính vô hướng của lớp Students được ánh xạ vào cơ sở dữ liệu

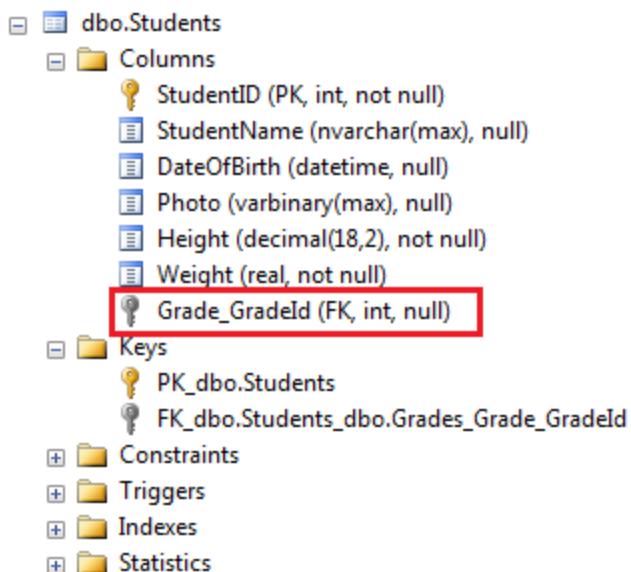
- *Thuộc tính điều hướng*

Thuộc tính điều hướng biểu diễn một mối quan hệ từ thực thể này đến một thực thể khác. Có hai dạng thuộc tính điều hướng: Điều hướng kiểu tham chiếu (*Reference Navigation*) và điều hướng kiểu tập hợp (*Collection Navigation*). Các thuộc tính điều hướng đóng một vai trò quan trọng trong việc định nghĩa mối quan hệ giữa các thực thể.

Nếu một thực thể chứa một thuộc tính có kiểu là một thực thể khác thì thuộc tính đó được gọi là thuộc tính điều hướng tham chiếu. Nó trỏ đến một thực thể duy nhất và thể hiện bội số của một (1) trong các mối quan hệ thực thể một-nhiều. EF API sẽ tạo ra một cột khóa ngoại (*ForeignKey*) trong bảng cho những thuộc tính điều hướng tham chiếu đến một cột khóa chính (*PrimaryKey*) của một bảng khác trong cơ sở dữ liệu.

Chẳng hạn, trong ví dụ dưới đây, Grade là một thuộc tính điều hướng kiểu tham chiếu khi nó được định nghĩa trong lớp thực thể Student. Khi đó, EF API sẽ tạo một khóa ngoại là Grade_GradeId trong bảng Students. Cột này sẽ tham chiếu đến cột khóa chính GradeId trong bảng Grades (Hình 5.13).

```
public class Student
{
    // Danh sách các thuộc tính vô hướng
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[] Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }
    // Danh sách các thuộc tính điều hướng
    public Grade Grade { get; set; }
}
```



Hình 5.13. Thuộc tính điều hướng của lớp Students được ánh xạ thành cột khóa ngoại

- Thuộc tính điều hướng kiểu tập hợp

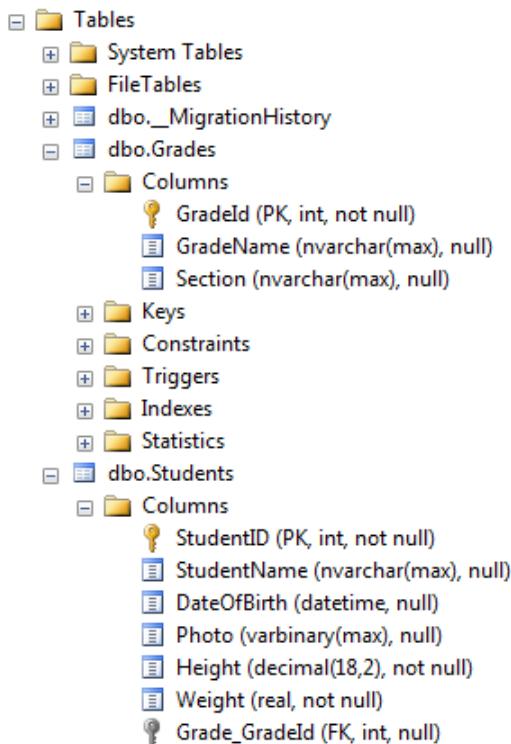
Nếu một thực thể chứa một thuộc tính có kiểu tập hợp, có thể chứa nhiều thực thể khác thì nó được gọi là thuộc tính điều hướng kiểu tập hợp. Nó thể hiện bộ số của nhiều (*) trong mối quan hệ một-nhiều. EF API không tạo thêm cột cho các thuộc tính dạng này trong các bảng ứng với lớp thực thể hiện tại. Tuy nhiên, nó sẽ tạo thêm cột khóa ngoại trong bảng tương ứng với thực thể chứa trong thuộc tính điều hướng kiểu tập hợp.

Chẳng hạn, trong ví dụ dưới đây, thực thể Grade chứa một thuộc tính điều hướng kiểu tập hợp là `ICollection<Student>`. EF sẽ không tạo thêm cột nào trong bảng Grades. Tuy nhiên, ở đây, thực thể Student được chỉ định là kiểu của các phần tử trong tập hợp. Vì thế, EF sẽ tạo một cột `Grade_GradeId` trong bảng Students trong cơ sở dữ liệu (Hình 5.14).

```
public class Grade
{
    public int GradeId { get; set; } // Mã lớp
```

```
public string GradeName { get; set; } // Tên lớp
public string Section { get; set; } // Ban

public ICollection<Student> Students { get; set; } // Thuộc tính điều
hướng kiểu tập hợp
}
```



Hình 5.14. Minh họa bảng Grades và Students trong cơ sở dữ liệu

5.6.9. Các loại thực thể trong Entity Framework

Có hai loại thực thể trong Entity Framework là POCO Entity và Dynamic Proxy Entity. POCO Entities – hay còn gọi là các đối tượng CLR đơn giản – là một lớp không phụ thuộc hay kế thừa bất kỳ lớp cơ sở của một framework cụ thể nào. Nó giống như bất kỳ lớp CLR bình thường khác trong .NET. Loại thực thể này được hỗ trợ trong cả EF6 và EF Core. Các thực thể POCO hỗ trợ hầu hết các thao tác truy vấn, thêm, cập nhật, xóa dữ liệu giống như các thực thể được sinh ra bởi mô hình dữ liệu thực thể EDM. Ví dụ sau cho thấy một thực thể POCO tên là Student.

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[] Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }
```

```
public StudentAddress StudentAddress { get; set; }  
public Grade Grade { get; set; }  
}
```

Dynamic Proxy là một lớp trung gian được tạo ra tại thời điểm thực thi để chứa thực thể POCO. Các thực thể Dynamic Proxy cho phép sử dụng tính năng tải dữ liệu khi cần (*lazy loading*). Loại thực thể này chỉ được hỗ trợ trong EF6. EF Core 2 vẫn chưa hỗ trợ dạng này. Một thực thể POCO phải hội đủ các yêu cầu sau đây mới trở thành một POCO Proxy:

- Lớp POCO phải được khai báo với phạm vi public;
- Lớp POCO không được khai báo với từ khóa sealed;
- Lớp POCO không phải là lớp trừu tượng;
- Thuộc tính điều hướng phải được khai báo với 2 từ khóa public và virtual;
- Mỗi thuộc tính điều hướng kiểu tập hợp phải khai báo với kiểu ICollection<T>;
- Tùy chọn ProxyCreationEnabled trong lớp ngũ cẩm không được gán bằng giá trị false.

Chẳng hạn, thực thể POCO có tên Student dưới đây thỏa mãn tất cả các yêu cầu trên, vì thế, nó có thể trở thành một thực thể proxy động tại thời điểm chạy chương trình.

```
public class Student  
{  
    public int StudentID { get; set; }  
    public string StudentName { get; set; }  
    public DateTime? DateOfBirth { get; set; }  
    public byte[] Photo { get; set; }  
    public decimal Height { get; set; }  
    public float Weight { get; set; }  
  
    public virtual StudentAddress StudentAddress { get; set; }  
    public virtual Grade Grade { get; set; }  
}
```

Theo mặc định, tính năng proxy động được kích hoạt cho mọi thực thể. Tuy nhiên, chúng ta có thể tắt tính năng này bằng cách gán context.Configuration.ProxyCreationEnabled = false. Tại thời điểm thực thi, EF API sẽ tạo ra một thể hiện của lớp proxy động tương ứng với thực thể Student ở trên. Trong trường hợp này, kiểu của lớp proxy động ứng với thực thể Student sẽ là: System.Data.Entity.DynamicProxies.Student (Hình 5.15).

```
using (SchoolDBEntities context = new SchoolDBEntities())
{
    Student studentEntity = context.Students.FirstOrDefault<Student>();
}
```

A screenshot of a C# code editor. The variable `studentEntity` is highlighted with a red rectangle. A tooltip or status bar at the bottom shows the full path of the proxy object: `studentEntity {System.Data.Entity.DynamicProxies.Student_8C9C606A94AFB4EB28256E8214C30B620217205B13ED3916324B4A}`.

Hình 5.15. Thể hiện của lớp proxy động với thực thể Student

Để tìm ra kiểu của đối tượng được bọc trong kiểu proxy động, ta có thể sử dụng phương thức `ObjectContext.GetObjectType()` như được minh họa trong hình 5.16:

```
using (SchoolDBEntities context = new SchoolDBEntities())
{
    Student studentEntity = context.Students.FirstOrDefault<Student>();
    var entityType = ObjectContext.GetObjectType(studentEntity.GetType());
}
```

A screenshot of a C# code editor. The variable `entityType` is highlighted with a red rectangle. A tooltip or status bar at the bottom shows the type information: `entityType {Name = "Student" FullName = "EFBasicTutorials.Student"}`.

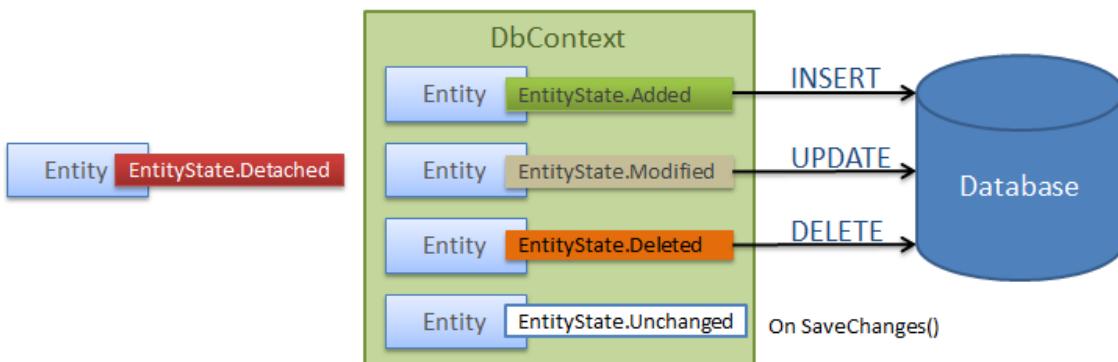
Hình 5.16. Kiểu đối tượng bọc của lớp proxy động với thực thể Student

5.6.10. Trạng thái của thực thể

EF API quản lý trạng thái của mỗi thực thể trong suốt thời gian sống của chúng. Mỗi thực thể có một trạng thái dựa vào thao tác được thực hiện trên thực thể đó thông qua lớp ngũ cảnh. Trạng thái của thực thể được biểu diễn bởi kiểu enum `System.Data.Entity.EntityState` trong EF6 và `Microsoft.EntityFrameworkCore.EntityState` trong EF Core. Enum này chứa những giá trị sau: `Added`, `Modified`, `Deleted`, `Unchanged`, `Detached`

Lớp ngũ cảnh không chỉ lưu giữ tham chiếu tới mọi đối tượng thực thể ngay khi đọc chúng lên từ cơ sở dữ liệu mà còn lưu vết trạng thái của các thực thể đó và quản lý các thay đổi được thực hiện trên các thuộc tính của thực thể. Tính năng này còn được gọi là *theo dõi các thay đổi* (Change Tracking). Thay đổi trạng thái của thực thể từ `Unchanged` sang `Modified` được xử lý tự động bởi lớp ngũ cảnh. Tất cả các thay đổi khác phải được thực hiện tường minh thông qua các phương thức thích hợp của lớp `DbContext` hoặc `DbSet`.

EF API xây dựng và thực thi các lệnh `INSERT`, `UPDATE`, `DELETE` dựa trên trạng thái của một thực thể khi phương thức `context.SaveChanges()` được gọi. Nó sẽ thực thi câu lệnh `INSERT` đối với những thực thể đang mang trạng thái là `Added`. Lệnh `UPDATE` sẽ được thực thi nếu trạng thái của thực thể là `Modified` và nếu trạng thái thực thể là `Deleted` thì lệnh `DELETE` sẽ được thực hiện. Lớp ngũ cảnh không theo dõi các thực thể đang mang trạng thái `Detached`. Trạng thái của thực thể đóng một vai trò quan trọng trong Entity Framework. Hình sau đây minh họa tầm quan trọng của chúng (Hình 5.17).

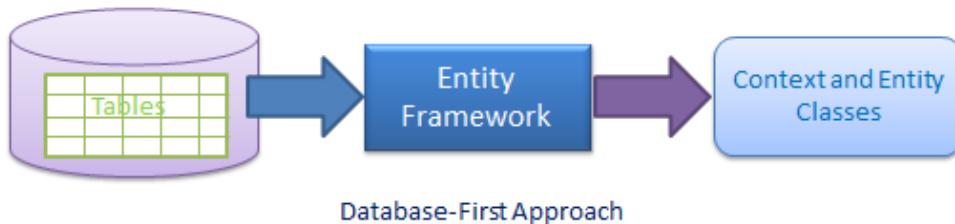


Hình 5.17. Các trạng thái của thực thể

5.6.11. Các hướng tiếp cận đối với Entity Framework

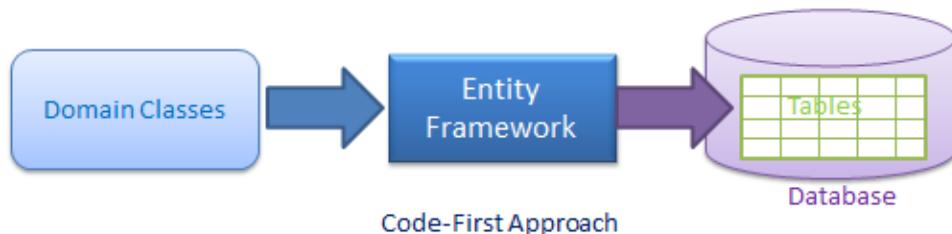
Có 3 hướng tiếp cận khác nhau mà chúng ta có thể sử dụng để phát triển ứng dụng với Entity Framework:

- *Database-First*: Đối với hướng tiếp cận này, chúng ta tạo ra lớp ngữ cảnh và các thực thể từ cơ sở dữ liệu có sẵn sử dụng trình EDM Wizard tích hợp sẵn trong Visual Studio hoặc thực thi các lệnh của EF. EF6 hỗ trợ đầy đủ các tính năng đối với phương pháp này (Hình 5.18). Tuy nhiên, EF Core còn có một số giới hạn.



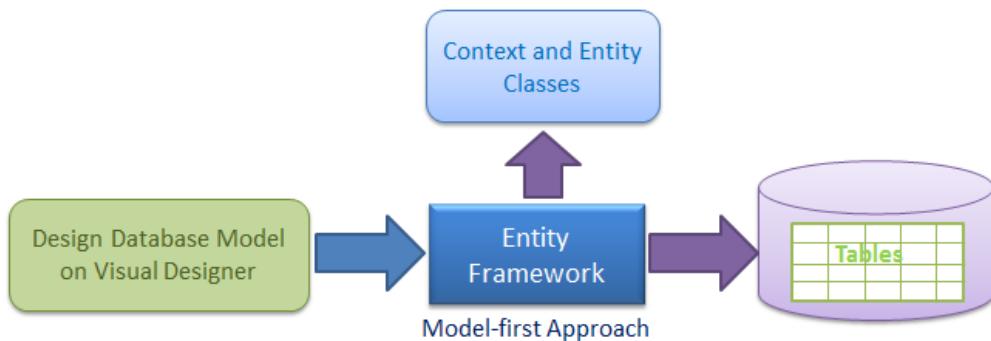
Hình 5.18. Cách tiếp cận Database - First

- *Code-First*: Hướng tiếp cận này được sử dụng khi chưa có sẵn một cơ sở dữ liệu cho ứng dụng. Trong trường hợp này, bắt đầu với việc viết mã để tạo ra các thực thể (hay các lớp trong miền ứng dụng) và lớp ngữ cảnh trước. Sau đó sử dụng các lệnh tích hợp của EF để tạo ra cơ sở dữ liệu từ những lớp này. Các lập trình viên tuân thủ nguyên lý DDD (*Domain-Driven Design*) thích bắt đầu với việc viết mã trên các lớp miền ứng dụng trước và rồi tạo cơ sở dữ liệu cần thiết để lưu trữ dữ liệu (Hình 5.19).



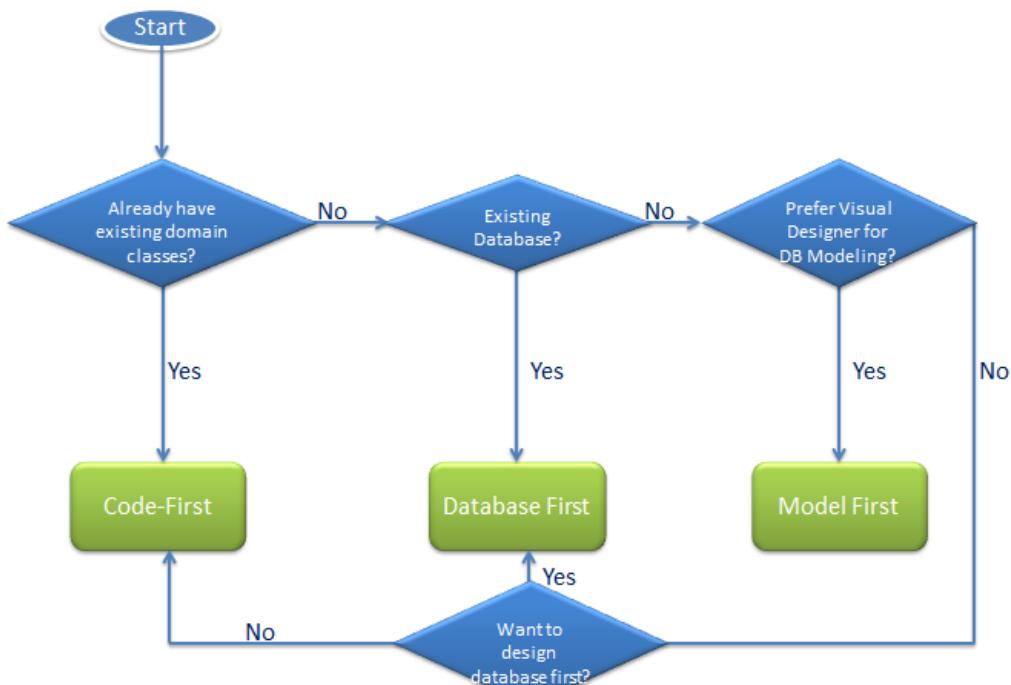
Hình 5.19. Cách tiếp cận Code - First

- **Model-First:** Đối với hướng tiếp cận này, trước hết cần thiết kế các thực thể, các mối quan hệ và sự phân cấp kế thừa trực tiếp trên màn hình trực quan của Visual Studio, sau đó hệ thống tự động sinh ra các lớp thực thể, lớp ngũ cành và các đoạn mã để sinh cơ sở dữ liệu từ mô hình trực quan đó (Hình 5.20). EF6 hỗ trợ hướng tiếp cận này nhưng có nhiều điểm hạn chế trong khi EF Core không hỗ trợ phương pháp này.



Hình 5.20. Cách tiếp cận Model - First

Để lựa chọn một hướng tiếp cận đúng đắn cho việc phát triển ứng dụng của mình dùng Entity Framework, người đọc có thể tham khảo lược đồ trong Hình 5.21. Theo đó, nếu ứng dụng của đã có sẵn các lớp nghiệp vụ thì nên sử dụng phương pháp Code-First vì có thể tạo ra một cơ sở dữ liệu từ những lớp hiện có. Nếu đang có sẵn một cơ sở dữ liệu thì có thể tạo ra một mô hình dữ liệu thực thể EDM từ cơ sở dữ liệu đó bằng phương pháp Database-First. Nếu chưa có cả cơ sở dữ liệu và các lớp nghiệp vụ và muốn thiết kế mô hình cơ sở dữ liệu bằng công cụ trực quan thì hướng tiếp cận Model-First sẽ là một lựa chọn phù hợp.



Hình 5.21. Lược đồ phỏng cách lựa chọn hướng tiếp cận

5.6.12. Các tình huống lưu trữ trong Entity Framework

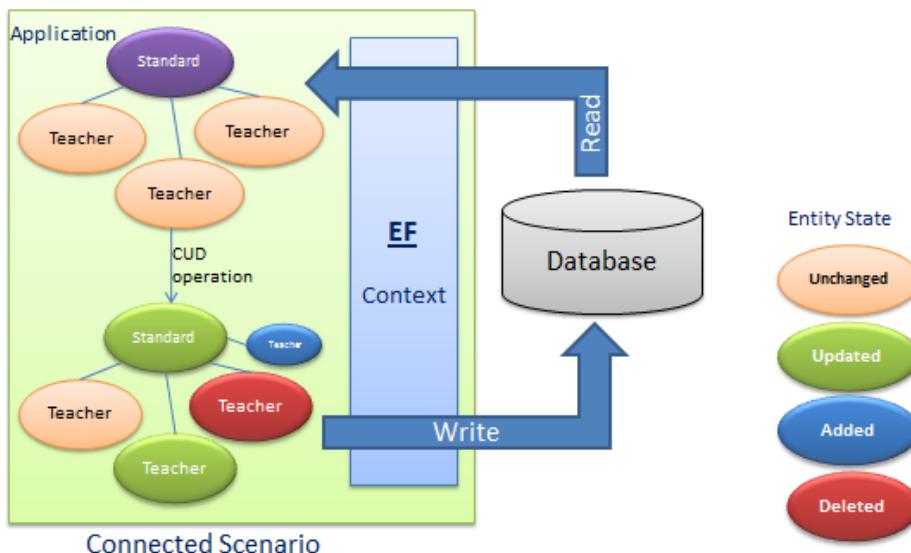
Có hai tình huống lưu trữ một thực thể vào cơ sở dữ liệu sử dụng Entity Framework: Kết nối (*Connected Scenario*) và không kết nối (*Disconnected Scenario*).

- *Tình huống có kết nối*

Trong trường hợp này, cùng một thê hiện của lớp ngũ cảnh (được kế thừa từ lớp DbContext) được sử dụng cho cả việc đọc và lưu các thực thể. Nó theo dõi tất cả các thực thể trong suốt thời gian sống của chúng. Phương pháp này rất hữu ích trong các ứng dụng Windows Form có kết nối tới một cơ sở dữ liệu cục bộ hoặc một cơ sở dữ liệu nào đó trong cùng mạng. Ưu và nhược điểm của tình huống này được thống kê trong Bảng 5.5, Hình 5.22 minh họa tính huống này.

Bảng 5.5. Ưu và nhược điểm của tình huống có kết nối

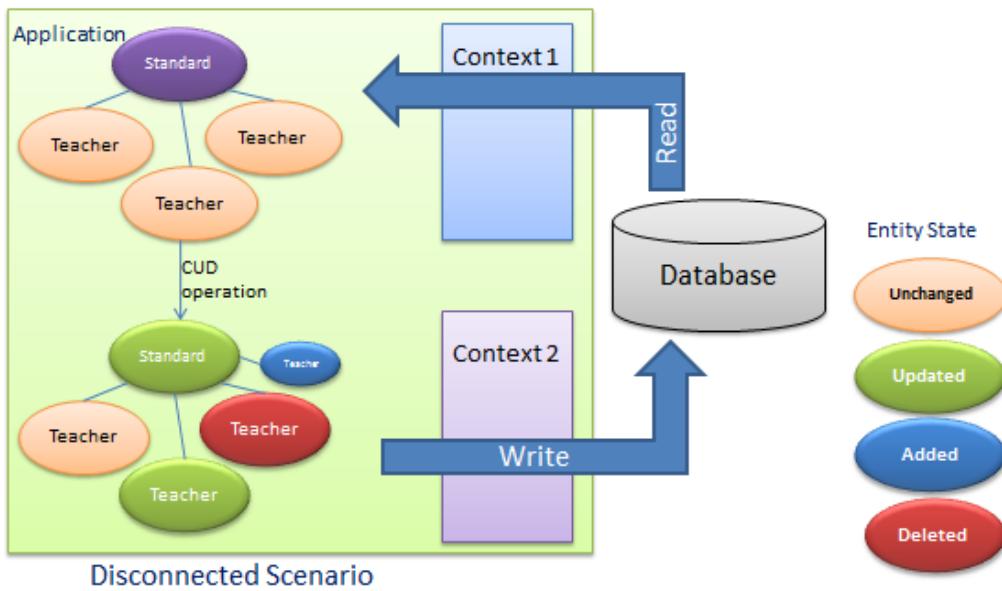
Ưu điểm	Nhược điểm
<ul style="list-style-type: none"> - Thực thi các lệnh nhanh chóng; - Đổi tượng context theo dõi tất cả các thực thể và tự động gán trạng thái thích hợp khi các thay đổi xảy ra trên các thực thể đó. 	<ul style="list-style-type: none"> - Đổi tượng context luôn sống vì thê kết nối tới cơ sở dữ liệu luôn ở trạng thái mở (Open); - Tiêu tốn nhiều tài nguyên.



Hình 5.22. Minh họa tình huống có kết nối

- *Tình huống không kết nối*

Đối với tình huống này, các thê hiện khác nhau của lớp ngũ cảnh được dùng để truy vấn và lưu trữ các thực thể vào cơ sở dữ liệu. Một thê hiện của lớp ngũ cảnh sẽ bị hủy sau khi truy vấn và đọc dữ liệu. Một thê hiện mới được tạo ra để lưu trữ các thực thể vào cơ sở dữ liệu (Hình 5.23). Tình huống không kết nối thường phức tạp bởi vì thê hiện của lớp ngũ cảnh không theo dõi thay đổi trên các thực thể. Vì thế, ta phải gán một trạng thái thích hợp cho mỗi thực thể trước khi lưu chúng vào cơ sở dữ liệu dùng phương thức `context.SaveChanges()`.



Hình 5.23. Minh họa tình huống không kết nối

Trong Hình 5.23, ứng dụng đọc các thực thể dùng context1 rồi thực hiện một vài lệnh thêm, cập nhật, xóa sử dụng context2. Trong tình huống này, context2 không hề biết thao tác nào đã được thực hiện trên các thực thể. Bảng 5.6 thống kê một số ưu và nhược điểm của tình huống có kết nối.

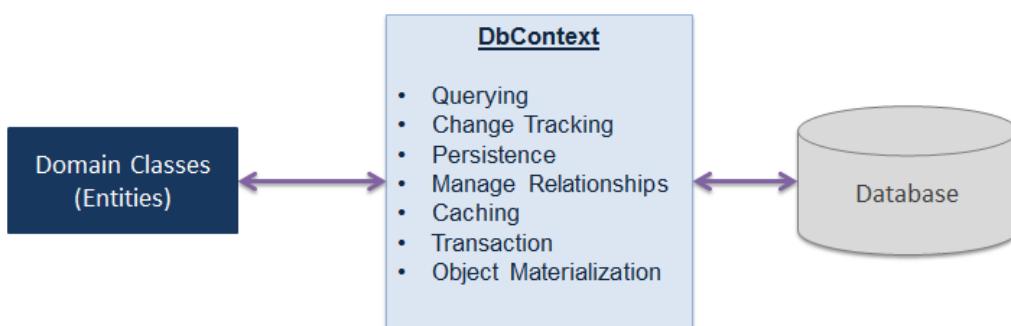
Bảng 5.6. Ưu và nhược điểm của tình huống có kết nối

Ưu điểm	Nhược điểm
<ul style="list-style-type: none"> Sử dụng ít tài nguyên hơn so với tình huống có kết nối ở trên; Không cần giữ kết nối luôn mở. 	<ul style="list-style-type: none"> Cần phải gán trạng thái phù hợp cho các thực thể trước khi lưu chúng; Thực thi chậm hơn so với tình huống có kết nối.

Phương pháp này hữu ích trong các ứng dụng web hoặc các ứng dụng có kết nối tới một cơ sở dữ liệu ở xa.

5.6.13. Lớp DbContext

Lớp DbContext là một lớp quan trọng trong Entity Framework. Nó chính là cầu nối giữa các lớp thực thể với cơ sở dữ liệu. Lớp DbContext là lớp chịu trách nhiệm chính trong việc tương tác với cơ sở dữ liệu, bao gồm các hoạt động sau đây (Hình 5.24).



Hình 5.24. Các chức năng của lớp DbContext

- *Truy vấn:* Chuyển đổi các truy vấn LINQ-to-Entities thành các truy vấn SQL và gửi chúng tới cơ sở dữ liệu để thực thi câu lệnh truy vấn;
- Theo dõi các thay đổi xảy ra trên các thực thể sau khi được truy vấn từ cơ sở dữ liệu;
- *Lưu trữ dữ liệu:* Thực hiện các thao tác INSERT, UPDATE và DELETE dựa trên trạng thái của các thực thể;
- *Caching:* cung cấp tính năng lưu trữ tạm vào bộ đệm. DbContext lưu trữ các thực thể đã được đọc lên từ cơ sở dữ liệu trong suốt thời gian sống;
- Quản lý mối quan hệ sử dụng CSDL, MSL và SSDL trong hướng tiếp cận Database-First hoặc Model-First và sử dụng các cấu hình API linh hoạt trong hướng tiếp cận Code-First;
- Chuyển đổi dữ liệu thô từ cơ sở dữ liệu thành các đối tượng trong ứng dụng;

Chương trình 5.8 cho thấy ví dụ về lớp SchoolDBEntities kế thừa từ lớp DbContext. Lớp này chứa các thuộc tính có kiểu DbSet< TEntity > để biểu diễn mọi thực thể cùng kiểu TEntity. Nó cũng chứa các phương thức tương ứng với các thủ tục và các thuộc tính tương ứng với các khung nhìn (View). Phương thức OnModelCreating cho phép chúng ta có thể cấu hình mô hình EDM sử dụng lớp DbModelBuilder sử dụng Fluent API của EF6. Một số phương thức và thuộc tính quan trọng của lớp DbContext được nêu trong Bảng 5.7 và Bảng 5.8.

Chương trình 5.8. Lớp SchoolDBEntities kế thừa lớp DbContext định nghĩa các thủ tục

```
namespace EFTutorials
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Data.Entity.Core.Objects;
    using System.Linq;

    public partial class SchoolDBEntities : DbContext
    {
        public SchoolDBEntities()
            : base("name=SchoolDBEntities")
        {
        }

        protected override void OnModelCreating(
            DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Course> Courses { get; set; }
        public virtual DbSet<Standard> Standards { get; set; }
        public virtual DbSet<Student> Students { get; set; }
    }
}
```

```
public virtual DbSet<StudentAddress> StudentAddresses {get;set;}
public virtual DbSet<Teacher> Teachers { get; set; }
public virtual DbSet<View_StudentCourse> View_StudentCourse
{ get; set; }

public virtual ObjectResult<GetCoursesByStudentId_Result>
GetCoursesByStudentId(Nullable<int> studentId)
{
    var studentIdParameter = studentId.HasValue
        ? new ObjectParameter("StudentId", studentId)
        : new ObjectParameter("StudentId", typeof(int));

    return ((IObjectContextAdapter)this)
        .ObjectContext.ExecuteFunction<GetCoursesByStudentId_Result>(
            "GetCoursesByStudentId", studentIdParameter);
}

public virtual int sp_DeleteStudent(Nullable<int> studentId)
{
    var studentIdParameter = studentId.HasValue
        ? new ObjectParameter("StudentId", studentId)
        : new ObjectParameter("StudentId", typeof(int));

    return ((IObjectContextAdapter)this)
        .ObjectContext.ExecuteFunction(
            "sp_DeleteStudent", studentIdParameter);
}

public virtual ObjectResult<Nullable<decimal>>
sp_InsertStudentInfo(Nullable<int> standardId, string studentName)
{
    var standardIdParameter = standardId.HasValue ?
        new ObjectParameter("StandardId", standardId) :
        new ObjectParameter("StandardId", typeof(int));

    var studentNameParameter = studentName != null ?
        new ObjectParameter("StudentName", studentName) :
        new ObjectParameter("StudentName", typeof(string));

    return ((IObjectContextAdapter)this)
        .ObjectContext.ExecuteFunction<Nullable<decimal>>(
            "sp_InsertStudentInfo",
            standardIdParameter, studentNameParameter);
}

public virtual int sp_UpdateStudent(Nullable<int> studentId,
    Nullable<int> standardId, string studentName)
{
    var studentIdParameter = studentId.HasValue ?
        new ObjectParameter("StudentId", studentId) :
        new ObjectParameter("StudentId", typeof(int));
```

```

        var standardIdParameter = standardId.HasValue ?
            new ObjectParameter("StandardId", standardId) :
            new ObjectParameter("StandardId", typeof(int));

        var studentNameParameter = studentName != null ?
            new ObjectParameter("StudentName", studentName) :
            new ObjectParameter("StudentName", typeof(string));

        return ((IObjectContextAdapter)this)
            .ObjectContext.ExecuteFunction("sp_UpdateStudent",
                studentIdParameter,
                standardIdParameter, studentNameParameter);
    }
}
}

```

Bảng 5.7. Các phương thức quan trọng của lớp DbContext

Phương thức	Mục đích sử dụng
Entry	Lấy ra đối tượng EbEntityEntry ứng với một thực thể cho trước. Đối tượng này cung cấp khả năng truy xuất tới các thông tin theo dõi các thay đổi bên trong thực thể và các thao tác đối với thực thể đó.
SaveChanges	Thực thi các câu lệnh INSERT, UPDATE, DELETE tới cơ sở dữ liệu đối với các thực thể có trạng thái tương ứng Added, Modified và Deleted.
SaveChangesAsync	Giống như phương thức SaveChanges nhưng thực thi bất đồng bộ.
Set	Tạo một đối tượng DbSet< TEntity > để thực hiện truy vấn và lưu trữ các đối tượng có kiểu TEntity.
OnModelCreating	Dùng để cấu hình mô hình EDM.

Bảng 5.8. Các thuộc tính quan trọng của lớp DbContext

Thuộc tính	Mục đích sử dụng
ChangeTracker	Cung cấp khả năng truy xuất tới thông tin và các thao tác đối với các thực thể mà đối tượng context đang theo dõi.
Configuration	Cung cấp khả năng truy xuất tới các tùy chọn cấu hình.
Database	Cung cấp khả năng truy xuất tới thông tin và các thao tác liên quan tới cơ sở dữ liệu.

5.6.14. Lớp DbSet

Lớp DbSet biểu diễn một tập các thực thể. Nó được sử dụng cho các thao tác thêm mới, đọc, cập nhật và xóa các thực thể. Lớp ngữ cảnh (kế thừa từ lớp DbContext) phải chứa các thuộc tính có kiểu DbSet đối với các thực thể ánh xạ đến các bảng và khung nhìn trong cơ sở dữ liệu. Bảng 5.9 trình bày một số phương thức quan trọng của lớp này, Hình 5.25 biểu diễn một số thực thể khi lưu trữ bằng DbSet.

Bảng 5.9. Các phương thức chính của lớp DbSet

Phương thức	Kiểu trả về	Mô tả
Add(TEntity entity)	TEntity	<p>Thêm thực thể cho trước vào ngũ cảnh và gán trạng thái là Added. Khi các thay đổi được lưu, các thực thể ở trạng thái Added sẽ được thêm vào cơ sở dữ liệu. Sau khi đã lưu, trạng thái của đối tượng thay đổi thành Unchanged.</p> <p>Ví dụ: context.Students.Add(studentEntity);</p>
Attach(TEntity entity)	TEntity	<p>Đưa thực thể cho trước vào đối tượng context nhưng đặt ở trạng thái Unchanged.</p> <p>Ví dụ: context.Students.Attach(studentEntity);</p>
Create()	TEntity	<p>Tạo một thể hiện mới của thực thể TEntity. Đối tượng trả về sẽ là một proxy nếu đối tượng context bên dưới được cấu hình là tạo đối tượng proxy và kiểu của thực thể thỏa mãn yêu cầu để tạo một proxy.</p> <p>Ví dụ: var newStudent = context.Students.Create();</p>
Find(object key)	TEntity	<p>Sử dụng giá trị trên cột khóa chính để tìm một thực thể được theo dõi bởi đối tượng context. Nếu thực thể chưa tồn tại, nó sẽ thực hiện một truy vấn và tìm dữ liệu trong dữ liệu nguồn. Hàm này trả về null nếu không tìm thấy thực thể.</p> <p>Ví dụ: var student = context.Students.Find(11);</p>
Include(string propSet)	DbQuery	<p>Trả về một truy vấn gộp chứa dữ liệu từ tập thực thể hiện tại và dữ liệu từ tập thực thể được bao gồm chỉ ra bởi tham số propSet.</p> <p>Ví dụ:</p> <pre>var stdList = context.Students.Include("Address") .ToList(); var stdList = context.Students.Include(s => s.Address) .ToList();</pre>
Remove(TEntity entity)	TEntity	<p>Đánh dấu thực thể cho trước là đã bị xóa bằng cách gán trạng thái của nó về Deleted. Khi các thay đổi được lưu, thực thể này sẽ bị xóa khỏi cơ sở dữ liệu. Thực thể này phải tồn tại trong đối tượng context và ở một trạng thái nào đó khác trước khi phương thức này được gọi.</p> <p>Ví dụ:</p> <pre>deletedStudent = context.Students.Remove(sv);</pre>
SqlQuery(string query)	DbSqlQuery	<p>Tạo một truy vấn thô để lấy về một tập thực thể có cùng kiểu. Theo mặc định, các thực thể trả về sẽ được theo dõi bởi đối tượng context.</p> <p>Ví dụ:</p> <pre>var student = context.Students.SqlQuery("SELECT * FROM Students WHERE Id = 1").FirstOrDefault();</pre>

```

EFBasicTutorials.SchoolDBEntities
namespace EFBasicTutorials
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Data.Entity.Core.Objects;
    using System.Linq;

    public partial class SchoolDBEntities : DbContext
    {
        public SchoolDBEntities()
            : base("name=SchoolDBEntities")
        {

        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Course> Courses { get; set; }
        public virtual DbSet<Standard> Standards { get; set; }
        public virtual DbSet<Student> Students { get; set; }
        public virtual DbSet<StudentAddress> StudentAddresses { get; set; }
        public virtual DbSet<Teacher> Teachers { get; set; }
        public virtual DbSet<View_StudentCourse> View_StudentCourse { get; set; }
    }
}

```

Hình 5.25. Một số thực thể trên C# khi lưu trữ bằng DbSet

5.6.15. Mối quan hệ giữa các thực thể trong EF6

Entity Framework hỗ trợ ba dạng quan hệ giữa các thực thể, giống như trong cơ sở dữ liệu, bao gồm: Quan hệ 1-1 (*One-to-One* hoặc *One-to-Zero*); Quan hệ 1-nhiều (*One-to-Many*); Quan hệ nhiều-nhiều (*Many-to-Many*). Sau đây là chi tiết các dạng quan hệ đó.

- *Quan hệ 1-1*

Hình 5.26 cho thấy hai thực thể Student và StudentAddress có mối quan hệ 1-1 (hoặc 1-0). Một sinh viên có thể có một địa chỉ hoặc không có địa chỉ. Entity Framework thể hiện mối kết hợp này bằng cách thêm một thuộc tính điều hướng dạng tham chiếu (*reference navigation property*) Student vào thực thể StudentEntity và ngược lại, thêm một thuộc tính điều hướng StudentAddress vào thực thể Student. Ngoài ra, thực thể StudentAddress sử dụng thuộc tính StudentId vừa là khóa chính vừa là khóa ngoại tham chiếu tới thực thể Student. Điều này tạo nên mối quan hệ 1-1 như trong đoạn mã sau đây:

```

public partial class Student
{
    public Student()
    {
        this.Courses = new HashSet<Course>();
    }
}

```

```

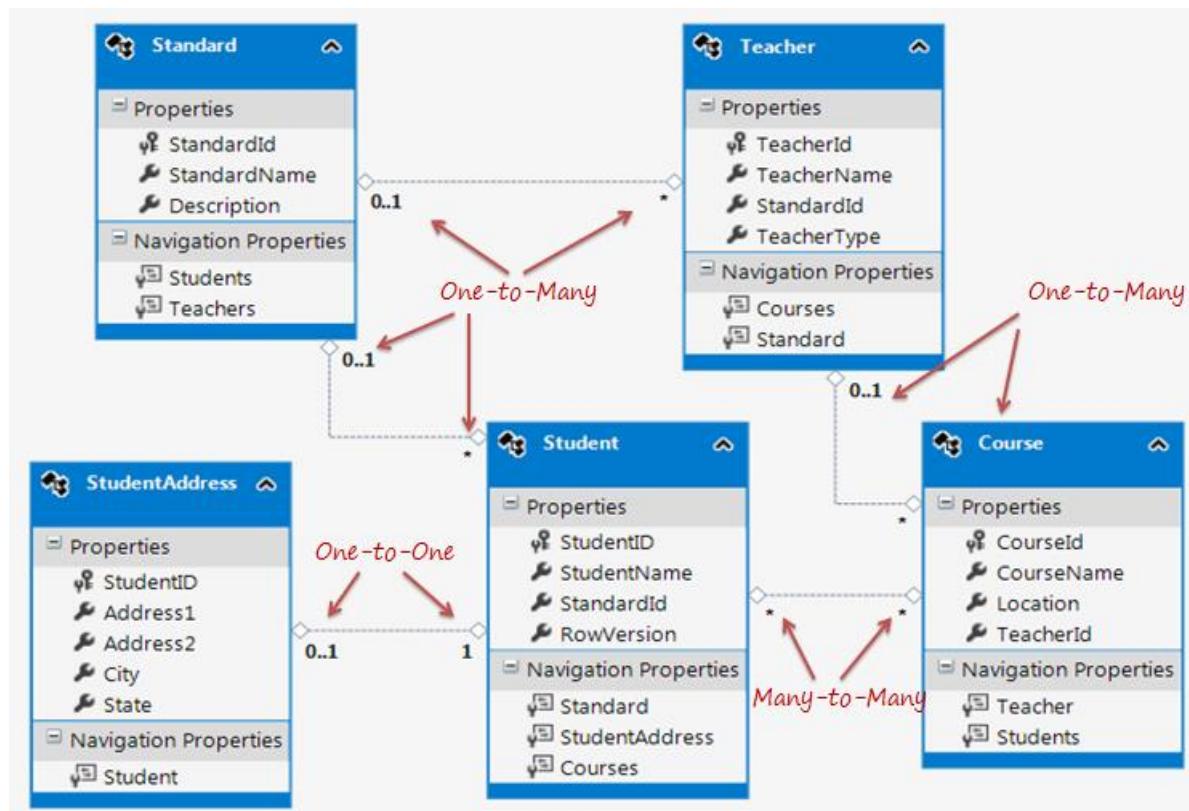
public int StudentID { get; set; }
public string StudentName { get; set; }
public Nullable<int> StandardId { get; set; }
public byte[] RowVersion { get; set; }

public virtual StudentAddress StudentAddress { get; set; } // Thuộc tính
kiểu tham chiếu tới StudentAddress
public virtual ICollection<Course> Courses { get; set; }
}

public partial class StudentAddress
{
    public int StudentID { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string State { get; set; }

    public virtual Student Student { get; set; } // Thuộc tính kiểu tham
    chiếu tới Student
}

```



Hình 5.26. Mối kết hợp giữa các thực thể trong EF6

- *Quan hệ 1-nhiều*

Các thực thể Standard và Teacher có một mối quan hệ 1-nhiều được đánh dấu bởi cặp bội số 1-* . Trong đó, * đại diện cho thành phần “nhiều” trong mối kết hợp. Điều

này có nghĩa là, một chức danh (Standard) có thể áp dụng cho nhiều giáo viên (Teacher) nhưng một giáo viên chỉ đi kèm một chức danh mà thôi.

Để biểu diễn mối quan hệ này, thực thể Standard chứa thuộc tính điều hướng dạng tập hợp (*collection navigation property*), có tên là Teachers, nhằm chỉ ra rằng chức năng có thể có một tập hợp các giáo viên (nhiều giáo viên) được phong chức danh đó. Ngược lại, thực thể Teacher có chứa một thuộc tính điều hướng Standard (thuộc tính tham chiếu) để chỉ ra rằng, một giáo viên được phong một chức danh. Ngoài ra, nó còn chứa một thuộc tính khóa ngoại là StandardId tham chiếu tới thực thể Standard. Điều này tạo nên mối kết hợp 1-nhiều như trong đoạn mã dưới đây:

```
public partial class Standard
{
    public Standard()
    {
        this.Teachers = new HashSet<Teacher>();
    }

    public int StandardId { get; set; }
    public string StandardName { get; set; }
    public string Description { get; set; }

    public virtual ICollection<Teacher> Teachers { get; set; } // Thuộc tính
    điều hướng kiểu tập hợp Icollection<Teacher>
}

public partial class Teacher
{
    public Teacher()
    {
        this.Courses = new HashSet<Course>();
    }

    public int TeacherId { get; set; }
    public string TeacherName { get; set; }
    public Nullable<int> TeacherType { get; set; }

    public Nullable<int> StandardId { get; set; }
    public virtual Standard Standard { get; set; } // Thuộc tính tham chiếu
    tới Standard
}
```

- *Quan hệ nhiều-nhiều*

Mỗi quan hệ dạng này được thể hiện giữa hai thực thể Student và Course và được đánh dấu bởi cặp bội số *-* . Điều đó có nghĩa là, một sinh viên có thể đăng ký nhiều khóa học và ngược lại, một khóa học có thể được giảng dạy cho nhiều sinh viên.

Trong cơ sở dữ liệu, sẽ có một bảng kết hợp StudentCourse chứa khóa chính của cả hai bảng Student và Course. Entity Framework biểu diễn mối quan hệ nhiều-nhiều

mà không cần dùng tới tập thực thể tương ứng với bảng kết hợp (*joining table*) như trong cơ sở dữ liệu. Thay vào đó, nó quản lý mối kết hợp thông qua việc ánh xạ (*mapping*).

Chẳng hạn, trong hình trên, thực thể Student chứa một thuộc tính điều hướng dạng tập hợp là Courses và ngược lại, thực thể Course có một thuộc tính điều hướng dạng tập hợp là Students để biểu diễn mối quan hệ nhiều-nhiều giữa chúng như trong đoạn mã sau đây:

```
public partial class Student
{
    public Student()
    {
        this.Courses = new HashSet<Course>();
    }

    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public Nullable<int> StandardId { get; set; }
    public byte[] RowVersion { get; set; }

    public virtual ICollection<Course> Courses { get; set; } // Thuộc tính
    điều hướng kiểu tập hợp
}

public partial class Course
{
    public Course()
    {
        this.Students = new HashSet<Student>();
    }

    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public System.Data.Entity.Spatial.DbGeography Location { get; set; }

    public virtual ICollection<Student> Students { get; set; } // Thuộc
    tính điều hướng kiểu tập hợp
}
```

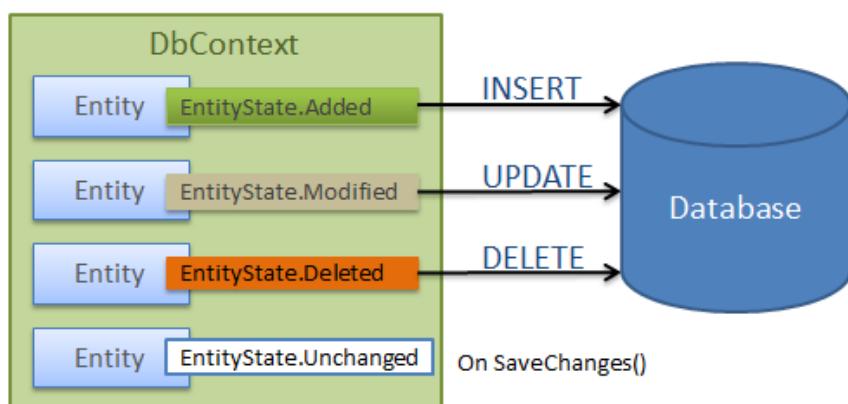
Lưu ý rằng, Entity Framework chỉ hỗ trợ quan hệ nhiều-nhiều chỉ khi bảng kết hợp không chứa thêm bất kỳ cột nào khác ngoài khóa chính của hai bảng thành phần trong mối kết hợp. Nếu bảng kết hợp có chứa thêm các cột khác, chẳng hạn như DateCreated, thì EDM sẽ tạo ra một thực thể cho bảng trung gian và bạn sẽ phải quản lý các thao tác đọc, thêm, xóa, cập nhật dữ liệu trong thực thể đó. Trong trường hợp này, mối quan hệ nhiều-nhiều được tách thành hai quan hệ 1-nhiều.

5.7. Lưu trữ dữ liệu trong tình huống có kết nối

Như đã đề cập trong Mục 5.6.12, trong Entity Framework, có hai tình huống lưu trữ dữ liệu khác nhau: Có kết nối và không kết nối. Trong trường hợp có kết nối, cùng

một đối tượng context được dùng cho cả việc lấy/đọc các thực thể từ cơ sở dữ liệu và lưu trữ chúng vào cơ sở dữ liệu. Trái lại, trong tình huống không kết nối, thực thể được đọc từ một đối tượng context và lưu trữ bởi một đối tượng context khác. Điều này dẫn đến việc có nhiều sự khác nhau giữa hai phương pháp lưu trữ dữ liệu. Trong phần này, ta sẽ học cách lưu trữ dữ liệu trong trường hợp có kết nối.

Việc lưu trữ dữ liệu của thực thể trong tình huống có kết nối tương đối đơn giản vì đối tượng context tự động theo dõi các thay đổi xảy ra trong các thực thể trong suốt thời gian sống của đối tượng context. Một thực thể với dữ liệu chưa trong các thuộc tính sẽ được thêm vào cơ sở dữ liệu, cập nhật hay xóa khỏi cơ sở dữ liệu tùy thuộc vào trạng thái của thực thể đó, còn gọi là EntityState.



Hình 5.27. Các trạng thái của thực thể và các lệnh SQL tương ứng

Hình 5.27 cho thấy Entity Framework xây dựng và thực thi các câu lệnh INSERT, UPDATE, DELETE cho các thực thể tùy thuộc vào trạng thái của thực thể là Added, Modified hay Deleted khi phương thức context.SaveChanges() được gọi. Vì trong trường hợp có kết nối, đối tượng context luôn theo dõi các thực thể, nó có thể tự động gán trạng thái EntityState thích hợp cho mỗi thực thể mỗi khi chúng được tạo, cập nhật hay xóa khỏi context.

5.7.1. Thêm dữ liệu mới

Phương thức DbSet.Add() được dùng để thêm một thực thể mới vào đối tượng context. Khi phương thức SaveChanges() được gọi, một mẫu tin mới tương ứng với thực thể đó được thêm vào cơ sở dữ liệu như trong đoạn mã sau đây:

```
using (var context = new SchoolDBEntities())
{
    var std = new Student()
    {
        FirstName = "Bill",
        LastName = "Gates"
    };
    context.Students.Add(std); // Thêm một đối tượng mới thực thể Student
```

```
        context.SaveChanges();}// Lưu thay đổi vào cơ sở dữ liệu  
    }
```

Trong ví dụ trên đây, lệnh context.Students.Add(std) sẽ thêm một thực thể Student mới vào đối tượng context và trạng thái của thực thể đó được gán bằng EntityState.Added. Phương thức context.SaveChanges() sẽ xây dựng và thực thi câu lệnh INSERT sau đây vào cơ sở dữ liệu.

```
exec sp_executesql N'INSERT [dbo].[Students]([FirstName], [LastName])  
VALUES (@0, @1)  
SELECT [StudentId]  
FROM [dbo].[Students]  
WHERE @@ROWCOUNT > 0 AND [StudentId] = scope_identity(),N  
' '@0 nvarchar(max) ,@1 nvarchar(max) ',@0=N'Bill',@1=N'Gates'  
go
```

5.7.2. Cập nhật dữ liệu

Trong tình huống có kết nối, EF API theo dõi mọi thực thể được đọc lên đối tượng context. Vì vậy, khi thay đổi dữ liệu trong thực thể, EF tự động đánh dấu trạng thái của thực thể là EntityState.Modified. Khi phương thức SaveChanges() được gọi, một câu lệnh UPDATE sẽ được tạo và thực thi trong cơ sở dữ liệu.

```
using (var context = new SchoolDBEntities())  
{  
    var std = context.Students.First<Student>(); // Tìm sinh viên đầu tiên  
    std.FirstName = "Steve"; } //Cập nhật FirstName của sinh viên tìm được là "Steve"  
    context.SaveChanges();} // Lưu thay đổi vào cơ sở dữ liệu  
}
```

Ở ví dụ này, ta truy vấn và lấy về mẫu tin Student đầu tiên trong danh sách dùng phương thức context.Students.First<Student>(). Ngay khi chúng ta thay đổi giá trị của FirstName, đối tượng context gán lại trạng thái của thực thể là EntityState.Modified vì các thay đổi được thực hiện trong phạm vi hoạt động của đối tượng context. Vì vậy, khi ta gọi hàm SaveChanges(), đối tượng context sẽ xây dựng và thực thi câu lệnh UPDATE sau đây trong cơ sở dữ liệu.

```
exec sp_executesql N'UPDATE [dbo].[Students]  
SET [FirstName] = @0  
WHERE ([StudentId] = @1)',  
N'@0 nvarchar(max) ,@1 int',@0=N'Steve',@1=2  
GO
```

Trong một câu lệnh UPDATE, EF API chỉ đưa vào các thuộc tính đã thay đổi giá trị, các thuộc tính khác bị bỏ qua. Chẳng hạn như ở trên, thuộc tính FirstName được cập nhật, vì thế, nó được đưa vào mệnh đề SET của câu lệnh UPDATE.

5.7.3. Xóa dữ liệu

Để xóa dữ liệu khỏi đối tượng context và cơ sở dữ liệu, ta dùng phương thức DbSet.Remove().

```
using (var context = new SchoolDBEntities())
{
    var std = context.Students.First<Student>();
    context.Students.Remove(std); // Xóa sinh viên std khỏi bảng
    Students
    context.SaveChanges(); // Lưu thay đổi vào cơ sở dữ liệu
}
```

Trong ví dụ trên, lời gọi hàm context.Students.Remove(std) chuyển trạng thái của thực thể std sang Deleted. Vì vậy, EF sẽ xây dựng và thực thi lệnh DELETE sau đây trong cơ sở dữ liệu.

```
exec sp_executesql N'DELETE [dbo].[Students]
WHERE ([StudentId] = @0)',N'@0 int',@0=1
Go
```

5.7.4. Truy vấn dữ liệu

EF có thể xây dựng và thực thi các câu lệnh truy vấn để lấy dữ liệu từ cơ sở dữ liệu bằng nhiều cách khác nhau. Những lệnh truy vấn này sau đó sẽ được chuyển thành các truy vấn SQL được thực thi trong cơ sở dữ liệu. EF hỗ trợ 3 kiểu truy vấn: LINQ-to-Entities; Entity SQL; và Native SQL (hay Raw SQL Query).

- *LINQ-to-Entities*

Language-Integrated Query (LINQ) là một ngôn ngữ truy vấn rất mạnh được giới thiệu lần đầu trong Visual Studio 2008. Các truy vấn LINQ-to-Entities hoạt động trên các tập thực thể (các thuộc tính có kiểu DbSet) để truy xuất tới dữ liệu bên dưới cơ sở dữ liệu. Chúng ta có thể sử dụng LINQ Method hoặc LINQ Syntax khi thực hiện truy vấn với EDM. Đoạn mã dưới đây cho thấy hai cách khác nhau để thực hiện một truy vấn LINQ-to-Entities để lấy dữ liệu từ bảng Student trong một cơ sở dữ liệu.

```
// Truy vấn LINQ-to-Entities sử dụng LinQ method
using (var context = new SchoolDBEntities())
{
    var query = context.Students
        .where(s => s.StudentName == "Bill")
        .FirstOrDefault<Student>();
}
```

```
//// Truy vấn LINQ-to-Entities sử dụng LINQ syntax
using (var context = new SchoolDBEntities())
{
    var query = from st in context.Students
                where st.StudentName == "Bill"
                select st;

    var student = query.FirstOrDefault<Student>();
}
```

Lớp DbSet thực thi giao diện Iqueryable nên chúng ta có thể sử dụng LINQ để truy vấn từ DbSet. Nó sẽ được chuyển đổi thành câu lệnh SQL. EF API thực thi truy vấn SQL này ở cơ sở dữ liệu bên dưới, sau đó lấy kết quả trả về và biến đổi nó thành các thực thể thích hợp trước khi trả kết quả về cho người dùng.

Giả sử SchoolDbEntities là lớp kế thừa từ DbContext và Students là một thuộc tính có kiểu DbSet trong lớp SchoolDbEntities. Khi đó, ta có thể dùng phương thức Find() của DbSet để tìm một thực thể dựa vào giá trị khóa chính của nó như sau.

```
var ctx = new SchoolDBEntities();
var student = ctx.Students.Find(1);
```

Câu lệnh ctx.Students.Find(1) trả về một mẫu tin chứa thông tin sinh viên có mã số là 1. Nếu không có mẫu tin nào được tìm thấy, kết quả trả về sẽ là null. Lệnh truy vấn trên được biến đổi thành truy vấn SQL như sau:

```
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
WHERE [Extent1].[StudentId] = @p0',N'@p0 int',@p0=1
go
```

Nếu muốn lấy một đối tượng Student đầu tiên có tên là Bill, ta có thể sử dụng phương thức First hoặc FirstOrDefault như sau:

```
using (var ctx = new SchoolDBEntities())
{
    var student = (from s in ctx.Students
                  where s.StudentName == "Bill"
                  select s).FirstOrDefault<Student>();
}
```

```
using (var ctx = new SchoolDBEntities())
{
    var student = ctx.Students
        .Where(s => s.StudentName == "Bill")
```

```
        .FirstOrDefault<Student>();  
    }
```

Câu lệnh SQL được xây dựng từ truy vấn LINQ ở trên:

```
SELECT TOP (1)  
[Extent1].[StudentID] AS [StudentID],  
[Extent1].[StudentName] AS [StudentName],  
[Extent1].[StandardId] AS [StandardId]  
FROM [dbo].[Student] AS [Extent1]  
WHERE 'Bill' = [Extent1].[StudentName]
```

EF cũng có thể xây dựng và thực thi các truy vấn SQL có tham số nếu truy vấn LINQ-to-Entites có sử dụng tham số như ví dụ sau:

```
using (var ctx = new SchoolDBEntities())  
{  
    string name = "Bill";  
    var student = ctx.Students  
        .Where(s => s.StudentName == name)  
        .FirstOrDefault<Student>();  
}
```

Lệnh truy vấn SQL tương ứng:

```
SELECT TOP (1)  
[Extent1].[StudentId] AS [StudentId],  
[Extent1].[Name] AS [Name]  
FROM [dbo].[Student] AS [Extent1]  
WHERE ([Extent1].[Name] = @p_linq_0) OR (([Extent1].[Name] IS NULL)  
    AND (@p_linq_0 IS NULL))',N'@p_linq_0  
nvarchar(4000)',@p_linq_0=N'Bill'
```

Để lấy danh sách tất cả các sinh viên có tên là Bill, ta viết truy vấn sử dụng phương thức `ToListAsync`, `ToArray`, `ToDictionary`, `ToLookup` như sau:

```
using (var ctx = new SchoolDBEntities())  
{  
    var  
studentList=ctx.Students.Where(s=>s.StudentName=="Bill").ToList();  
}
```

Lệnh truy vấn SQL tương ứng như sau:

```
SELECT  
[Extent1].[StudentID] AS [StudentID],  
[Extent1].[StudentName] AS [StudentName],  
[Extent1].[StandardId] AS [StandardId]  
FROM [dbo].[Student] AS [Extent1]
```

```
WHERE 'Bill' = [Extent1].[StudentName]
go
```

Toán tử group by hoặc phương thức GroupBy() được dùng khi muốn gom nhóm các mẫu tin trả về về theo một hoặc một số thuộc tính nào đó. Chẳng hạn, ví dụ dưới đây sẽ lấy về danh sách các nhóm sinh viên phân loại theo trình độ (*Standard*).

```
using (var ctx = new SchoolDBEntities())
{
    var students = from s in ctx.Students
                   group s by s.StandardId into studentsByStandard
                   select studentsByStandard;
    foreach (var groupItem in students) {
        Console.WriteLine(groupItem.Key);

        foreach (var stud in groupItem) {
            Console.WriteLine(stud.StudentId);
        }
    }
}
```

```
using (var ctx = new SchoolDBEntities())
{
    var students = ctx.Students.GroupBy(s => s.StandardId);

    foreach (var groupItem in students)
    {
        Console.WriteLine(groupItem.Key);

        foreach (var stud in groupItem)
        {
            Console.WriteLine(stud.StudentId);
        }
    }
}
```

Lệnh SQL tương ứng như sau:

```
SELECT
[Project2].[C1] AS [C1],
[Project2].[StandardId] AS [StandardId],
[Project2].[C2] AS [C2],
[Project2].[StudentID] AS [StudentID],
[Project2].[StudentName] AS [StudentName],
[Project2].[StandardId1] AS [StandardId1]
FROM (
    SELECT
        [Distinct1].[StandardId] AS [StandardId],
        1 AS [C1],
        [Extent2].[StudentID] AS [StudentID],
```

```
[Extent2].[StudentName] AS [StudentName],  
[Extent2].[StandardId] AS [StandardId1],  
CASE WHEN ([Extent2].[StudentID] IS NULL) THEN CAST(NULL AS int)  
ELSE 1 END AS [C2]  
FROM (SELECT DISTINCT  
    [Extent1].[StandardId] AS [StandardId]  
    FROM [dbo].[Student] AS [Extent1] ) AS [Distinct1]  
LEFT OUTER JOIN [dbo].[Student] AS [Extent2] ON  
([Distinct1].[StandardId] = [Extent2].[StandardId]) OR  
(([Distinct1].[StandardId] IS NULL) AND ([Extent2].[StandardId] IS  
NULL))  
) AS [Project2]  
ORDER BY [Project2].[StandardId] ASC, [Project2].[C2] ASC  
go
```

Khi muốn sắp xếp kết quả trả về theo một thứ tự nào đó, ta có thể dùng toán tử orderby cùng với từ khóa ascending/descending (LINQ Query syntax) hoặc phương thức OrderBy/OrderByDescending (LINQ Method syntax). Ví dụ sau đây cho thấy cách sắp xếp danh sách sinh viên theo tên:

```
using (var ctx = new SchoolDBEntities())  
{  
    var students = from s in ctx.Students  
                   orderby s.StudentName ascending  
                   select s;  
}
```

```
using (var ctx = new SchoolDBEntities())  
{  
    var students = ctx.Students.OrderBy(s => s.StudentName).ToList();  
    // or descending order  
    var descStudents = ctx.Students.OrderByDescending(s =>  
s.StudentName).ToList();  
}
```

Lệnh SQL tương ứng như sau:

```
SELECT  
[Extent1].[StudentID] AS [StudentID],  
[Extent1].[StudentName] AS [StudentName],  
[Extent1].[StandardId] AS [StandardId]  
FROM [dbo].[Student] AS [Extent1]  
ORDER BY [Extent1].[StudentName] ASC  
go
```

Các truy vấn LINQ to Entities không phải lúc nào cũng phải trả về các thực thể. Đôi khi, bạn chỉ muốn lấy giá trị của một vài thuộc tính từ các thực thể. EF hỗ trợ việc này bằng cách trả về một danh sách các đối tượng nặc danh (anonymous objects) hay các đối tượng không định kiểu. Chẳng hạn:

```
using (var ctx = new SchoolDBEntities())
{
    var anonymousObjResult = from s in ctx.Students
                                where s.StandardId == 1
                                select new {
                                    Id = st.StudentId,
                                    Name = st.StudentName
                                };

    foreach (var obj in anonymousObjResult)
    {
        Console.WriteLine(obj.Name);
    }
}
```

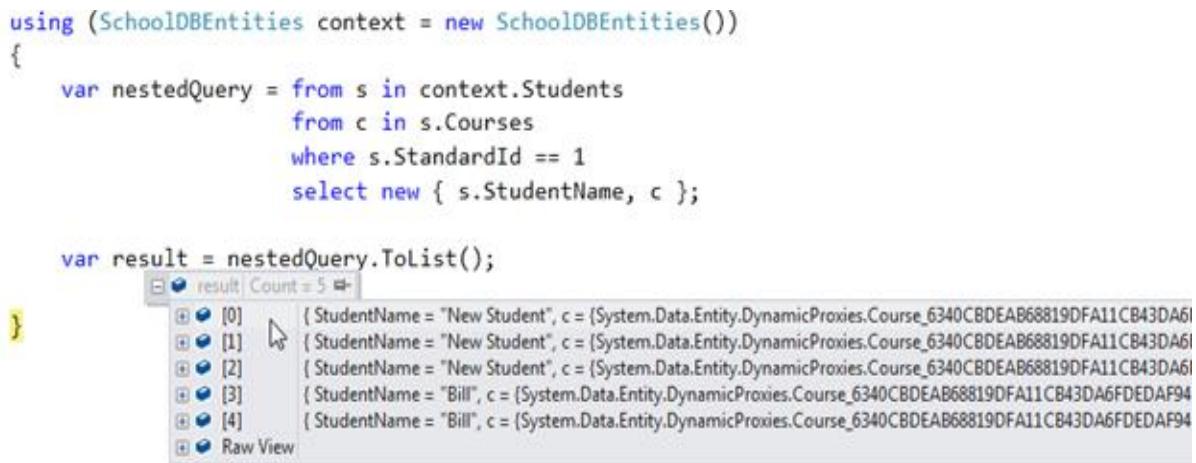
```
using (var ctx = new SchoolDBEntities())
{
    var anonymousObjResult = ctx.Students
                                .Where(st => st.Standard == 1)
                                .Select(st => new {
                                    Id = st.StudentId,
                                    Name = st.StudentName });

    foreach (var obj in anonymousObjResult)
    {
        Console.WriteLine(obj.Name);
    }
}
```

Ví dụ trên đây cho thấy một truy vấn trả về một danh sách các đối tượng nặc danh chứa các thuộc tính StudentId và StudentName. Ta gọi chúng là các đối tượng nặc danh vì không có lớp hay thực thể nào chứa các thuộc tính này. Trình biên dịch sẽ tự động tạo các lớp cần thiết để biểu diễn và đánh dấu các đối tượng đó là nặc danh. Khi truy vấn được thực thi, câu lệnh SQL sau đây sẽ được sinh ra và thực thi bên dưới cơ sở dữ liệu:

```
SELECT
[s].[StudentID] AS [Id], [s].[StudentName] AS [Name]
FROM [Student] AS [s]
WHERE [s].[StandardId] = 1
go
```

EF cũng cho phép chúng ta thực thi các truy vấn lồng nhau hoặc truy vấn có sự kết hợp giữa các tập thực thể khác nhau. Ví dụ, để lấy danh sách tên sinh viên đạt trình độ 1 và các khóa học mà sinh viên đó tham gia, ta viết lệnh truy vấn LINQ (soi kết quả bằng công cụ Debug như Hình 5.28).



The screenshot shows a debugger window with a list of five items. Each item contains a student name and a course object. The student names are "New Student" and "Bill". The course objects are represented as dynamic proxies with long GUID-like strings.

```
using (SchoolDBEntities context = new SchoolDBEntities())
{
    var nestedQuery = from s in context.Students
                      from c in s.Courses
                      where s.StandardId == 1
                      select new { s.StudentName, c };

    var result = nestedQuery.ToList();
}
```

Index	Student Name	Course Object
[0]	New Student	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA61}
[1]	New Student	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA61}
[2]	New Student	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA61}
[3]	Bill	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94}
[4]	Bill	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94}

Hình 5.28. Lệnh truy vấn LINQ khi soi bằng chức năng Debug

Kết quả của truy vấn trên là một danh sách các đối tượng nặc danh chứa hai thuộc tính tên sinh viên (StudentName) và khóa học (Course). Lệnh SQL tương ứng với truy vấn ở trên được tạo ra như sau:

```
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Join1].[CourseId1] AS [CourseId],
[Join1].[CourseName] AS [CourseName],
[Join1].[Location] AS [Location],
[Join1].[TeacherId] AS [TeacherId]
FROM [dbo].[Student] AS [Extent1]
INNER JOIN (SELECT [Extent2].[StudentId] AS [StudentId],
[Extent3].[CourseId] AS [CourseId1], [Extent3].[CourseName] AS [CourseName],
[Extent3].[Location] AS [Location], [Extent3].[TeacherId] AS [TeacherId]
FROM [dbo].[StudentCourse] AS [Extent2]
INNER JOIN [dbo].[Course] AS [Extent3]
ON [Extent3].[CourseId] = [Extent2].[CourseId] ) AS [Join1]
ON [Extent1].[StudentID] = [Join1].[StudentId]
WHERE 1 = [Extent1].[StandardId]
go
```

5.7.5. Eager loading

Eager loading là quá trình trong đó khi thực hiện một truy vấn trên một loại thực thể nào đó thì các thực thể liên quan cũng được nạp (tải) kèm theo như một phần của truy vấn. Khi đó, chúng ta không cần phải thực hiện một truy vấn khác để lấy các thực thể liên quan đó. Điều này có thể thực hiện được bằng cách sử dụng phương thức `Include()`.

Trong ví dụ dưới đây, truy vấn sẽ trả về danh sách sinh viên đi kèm với trình độ (xếp loại) của sinh viên đó bằng cách dùng phương thức `Include()`.

```
using (var context = new SchoolDBEntities())
{
    var stud1 = (from s in context.Students.Include("Standard")
                 where s.StudentName == "Bill"
                 select s).FirstOrDefault<Student>();
}
```

```
using (var ctx = new SchoolDBEntities())
{
    var stud1 = ctx.Students.Include("Standard")
        .Where(s => s.StudentName == "Bill").FirstOrDefault<Student>();
}
```

Khi đó, lệnh SQL được tạo ra sẽ có dạng như sau:

```
SELECT TOP (1)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent2].[StandardId] AS [StandardId],
[Extent2].[StandardName] AS [StandardName],
[Extent2].[Description] AS [Description]
FROM [dbo].[Student] AS [Extent1]
LEFT OUTER JOIN [dbo].[Standard] AS [Extent2] ON
[Extent1].[StandardId] = [Extent2].[StandardId]
WHERE 'Bill' = [Extent1].[StudentName]
```

Phương thức Include() cũng có thể nhận vào tham số là một biểu thức Lamda thay vì một chuỗi như trên. Để sử dụng được biểu thức Lamda trong phương thức Include, bạn cần nạp thêm namespace System.Data.Entity.

```
using System;
using System.Data.Entity;

class Program
{
    static void Main(string[] args)
    {
        using (var ctx = new SchoolDBEntities())
        {
            var stud1 = ctx.Students.Include(s => s.Standard)
                .Where(s => s.StudentName == "Bill")
                .FirstOrDefault<Student>();
        }
    }
}
```

Eager loading cũng hỗ trợ nạp (tải) các thực thể liên quan theo nhiều cấp. Chẳng hạn, ví dụ sau đây cho thấy cách lấy dữ liệu từ cả 3 tập thực thể Student, Standard và Teacher:

```
using (var ctx = new SchoolDBEntities())
{
    var stud1 = ctx.Students.Include("Standard.Teachers")
        .Where(s => s.StudentName == "Bill")
        .FirstOrDefault<Student>();
}
```

```
using (var ctx = new SchoolDBEntities())
{
    var stud1 = ctx.Students.Include(s => s.Standard.Teachers)
        .Where(s => s.StudentName == "Bill")
        .FirstOrDefault<Student>();
}
```

Câu truy vấn LINQ trên sẽ tạo ra câu lệnh SQL tương ứng như sau:

```
SELECT [Project2].[StudentID] AS [StudentID],
[Project2].[StudentName] AS [StudentName],
[Project2].[StandardId] AS [StandardId],
[Project2].[StandardName] AS [StandardName],
[Project2].[Description] AS [Description],
[Project2].[C1] AS [C1],
[Project2].[TeacherId] AS [TeacherId],
[Project2].[TeacherName] AS [TeacherName],
[Project2].[StandardId1] AS [StandardId1]
FROM (
    SELECT
        [Limit1].[StudentID] AS [StudentID],
        [Limit1].[StudentName] AS [StudentName],
        [Limit1].[StandardId1] AS [StandardId],
        [Limit1].[StandardName] AS [StandardName],
        [Limit1].[Description] AS [Description],
        [Project1].[TeacherId] AS [TeacherId],
        [Project1].[TeacherName] AS [TeacherName],
        [Project1].[StandardId] AS [StandardId1],
        CASE WHEN ([Project1].[TeacherId] IS NULL) THEN CAST(NULL AS int)
        ELSE 1 END AS [C1]
    FROM (SELECT TOP (1) [Extent1].[StudentID] AS [StudentID],
        [Extent1].[StudentName] AS [StudentName], [Extent1].[StandardId] AS [StandardId2],
        [Extent2].[StandardId] AS [StandardId1],
        [Extent2].[StandardName] AS [StandardName], [Extent2].[Description]
        AS [Description]
        FROM [dbo].[Student] AS [Extent1]
        LEFT OUTER JOIN [dbo].[Standard] AS [Extent2] ON
        [Extent1].[StandardId] = [Extent2].[StandardId]
        WHERE 'updated student' = [Extent1].[StudentName] ) AS
        [Limit1]
```

```
LEFT OUTER JOIN (SELECT  
    [Extent3].[TeacherId] AS [TeacherId],  
    [Extent3].[TeacherName] AS [TeacherName],  
    [Extent3].[StandardId] AS [StandardId]  
    FROM [dbo].[Teacher] AS [Extent3]  
    WHERE [Extent3].[StandardId] IS NOT NULL ) AS [Project1] ON  
[Limit1].[StandardId2] = [Project1].[StandardId]  
) AS [Project2]  
ORDER BY [Project2].[StudentID] ASC, [Project2].[StandardId] ASC,  
[Project2].[C1] ASC
```

5.7.6. Lazy loading

Trái ngược với *Eager loading*, *Lazy loading* không nạp dữ liệu liên quan ngay khi tải dữ liệu từ thực thể gốc. Các truy vấn để lấy về các thực thể liên quan sẽ được tạo và thực thi khi được yêu cầu (khi thuộc tính chứa các thực thể liên quan được sử dụng).

Ví dụ: Thực thể Student chứa thực thể StudentAddress. Trong Lazy loading, đối tượng context sẽ tải dữ liệu cho các thực thể Student từ cơ sở dữ liệu trước, sau đó, nó sẽ tải thực thể StudentAddress khi chúng ta truy xuất tới thuộc tính StudentAddress của đối tượng Student:

```
using (var ctx = new SchoolDBEntities())  
{  
    //Loading students only  
    IList<Student> studList = ctx.Students.ToList<Student>();  
  
    Student std = studList[0];  
  
    //Loads Student address for particular Student only (separate SQL  
    //query)  
    StudentAddress add = std.StudentAddress;  
}
```

Khi được thực thi, đầu tiên, câu lệnh SQL sau được thực thi để lấy về danh sách tất cả sinh viên:

```
SELECT  
[Extent1].[StudentID] AS [StudentID],  
[Extent1].[StudentName] AS [StudentName],  
[Extent1].[StandardId] AS [StandardId]  
FROM [dbo].[Student] AS [Extent1]
```

Sau đó, chương trình tiếp tục thực thi lệnh SQL bên dưới để lấy về thực thể StudentAddress:

```
exec sp_executesql N'SELECT  
[Extent1].[StudentID] AS [StudentID],  
[Extent1].[Address1] AS [Address1],
```

```
[Extent1].[Address2] AS [Address2],  
[Extent1].[City] AS [City],  
[Extent1].[State] AS [State]  
FROM [dbo].[StudentAddress] AS [Extent1]  
WHERE [Extent1].[StudentID] = @EntityKeyValue1,N'@EntityKeyValue1  
int',@EntityKeyValue1=1
```

Chúng ta có thể vô hiệu hóa tính năng Lazy loading trên một thực thể cụ thể hoặc toàn bộ đối tượng context. Để tắt tính năng Lazy loading cho một thuộc tính cụ thể, ta chỉ cần loại bỏ từ khóa virtual khỏi thuộc tính đó. Để tắt Lazy loading cho tất cả các thực thể trong đối tượng context, ta thiết lập giá trị false cho thuộc tính cấu hình LazyLoadingEnabled như sau:

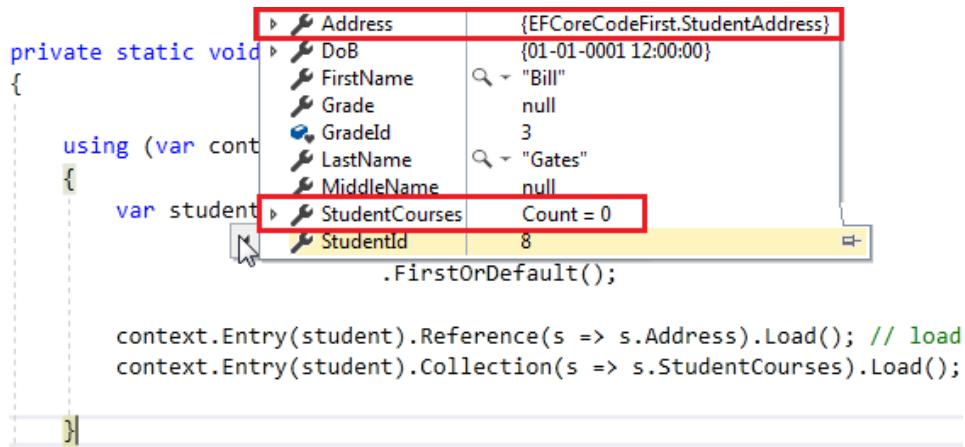
```
using System;  
using System.Data.Entity;  
using System.Data.Entity.Infrastructure;  
using System.Data.Entity.Core.Objects;  
using System.Linq;  
public partial class SchoolDBEntities : DbContext  
{  
    public SchoolDBEntities(): base("name=SchoolDBEntities")  
    {  
        this.Configuration.LazyLoadingEnabled = false;  
    }  
    protected override void OnModelCreating(DbModelBuilder  
modelBuilder)  
    {  
    }  
}
```

5.7.7. Explicit Loading

EF cũng cho phép chúng ta chỉ rõ các thực thể hay tập thực thể nào được tải khi được yêu cầu tại một thời điểm xác định. Thậm chí khi tính năng Lazy loading bị vô hiệu hóa, ta vẫn có thể sử dụng cách này để nạp dữ liệu của các thực thể liên quan khi cần thiết. Tuy nhiên, việc này cần được thực hiện một cách tường minh bằng cách sử dụng lời gọi hàm tới phương thức Load():

```
using (var context = new SchoolContext())  
{  
    var student = context.Students  
        .Where(s => s.FirstName == "Bill")  
        .FirstOrDefault<Student>();  
    // loads StudentAddress  
    context.Entry(student).Reference(s => s.StudentAddress).Load();  
    // loads Courses collection  
    context.Entry(student).Collection(s => s.StudentCourses).Load();  
}
```

Trong ví dụ trên, dòng lệnh context.Entry(student).Reference(s => s.StudentAddress).Load(); sẽ nạp thực thể StudentAddress. Phương thức Reference() được dùng để lấy đối tượng tham chiếu tới thuộc tính điều hướng StudentAddress và phương thức Load() chịu trách nhiệm tải dữ liệu. Kết quả được soi bằng chức năng Debug minh họa như trong Hình 5.29.



Hình 5.29. Lệnh tải dữ liệu khi soi bằng chức năng Debug

Tương tự, context.Entry(student).Collection(s => s.StudentCourses).Load(); nạp dữ liệu vào thuộc tính điều hướng Courses của thực thể Student. Phương thức Collections() lấy đối tượng biểu diễn thuộc tính điều hướng kiểu tập hợp Courses. Phương thức Load() thực thi câu lệnh SQL dưới cơ sở dữ liệu và lấy dữ liệu về rồi điền vào các đối tượng tham chiếu hay các thuộc tính kiểu tập hợp trong bộ nhớ.

Trong trường hợp chỉ muốn tải một phần của dữ liệu liên quan, chúng ta có thể trích lọc chúng trước khi tải bằng cách sử dụng phương thức Query() để cho phép viết thêm điều kiện truy vấn trên tập thực thể liên quan như trong mã nguồn sau đây:

```

using (var context = new SchoolContext())
{
    var student = context.Students
        .Where(s => s.FirstName == "Bill")
        .FirstOrDefault<Student>();

    context.Entry(student)
        .Collection(s => s.StudentCourses)
        .Query()
        .Where(sc => sc.CourseName == "Maths")
        .FirstOrDefault();
}

```

5.7.8. Thực thi một truy vấn SQL

EF cũng hỗ trợ việc thực thi trực tiếp một truy vấn SQL tới cơ sở dữ liệu bên dưới bằng cách sử dụng một trong ba phương thức sau: DbSet.SqlQuery(), DbContext.Database.SqlQuery() và DbContext.Database.ExecuteSqlCommand().

Phương thức DbSet.SqlQuery() được dùng để thực hiện một truy vấn SQL và trả về các thực thể. Chúng được theo dõi bởi đối tượng context giống như khi chúng ta thực hiện truy vấn LINQ. Chẳng hạn, ví dụ dưới đây cho thấy câu lệnh select * from Students được thực thi trong cơ sở dữ liệu và trả về danh sách tất cả các sinh viên. Các mẫu tin được chuyển đổi thành một danh sách các thực thể Student. Tên của các cột trong truy vấn SQL phải khớp với các thuộc tính của thực thể.

```
using (var ctx = new SchoolDBEntities())
{
    var studentList = ctx.Students
        .SqlQuery("Select * from Students")
        .ToList<Student>();
}
```

Chúng ta cũng có thể truyền các tham số vào lệnh SQL để làm điều kiện truy vấn bằng cách cung cấp đối tượng SqlParameter vào đối số của phương thức SqlQuery.

```
using (var ctx = new SchoolDBEntities())
{
    var student = ctx.Students
        .SqlQuery("Select * from Students where
StudentId=@id", new SqlParameter("@id", 1)).FirstOrDefault();
}
```

Lưu ý rằng, phương thức DbSet.SqlQuery chỉ thực hiện các truy vấn SQL trên bảng được ánh xạ tới thực thể được chỉ định và trả về dữ liệu lấy được từ bảng đó. Điều này có nghĩa là lệnh DbSet<Student>.SqlQuery() chỉ lấy dữ liệu từ bảng Students. Câu lệnh sau đây sẽ trả về lỗi:

```
using (var ctx = new SchoolDBEntities())
{
    //this will throw an exception
    var studentName = ctx.Students.SqlQuery("Select * from
Courses").ToList();
}
```

Phương thức Database.SqlQuery() linh động hơn vì cho phép trả về một giá trị có kiểu bất kỳ:

```
using (var ctx = new SchoolDBEntities())
{
    //Get student name of string type
    string studentName = ctx.Database.SqlQuery<string>("Select
studentname from Student where studentid=1").FirstOrDefault();
    //or
    string studentName = ctx.Database.SqlQuery<string>("Select
studentname from Student where studentid = @id", new
SqlParameter("@id", 1))
```

```
.FirstOrDefault();
}
```

Trong trường hợp muốn thực hiện các câu lệnh Insert, Update và Delete, phương thức Database.ExecuteNonQuery() sẽ là giải pháp hợp lý và hữu ích.

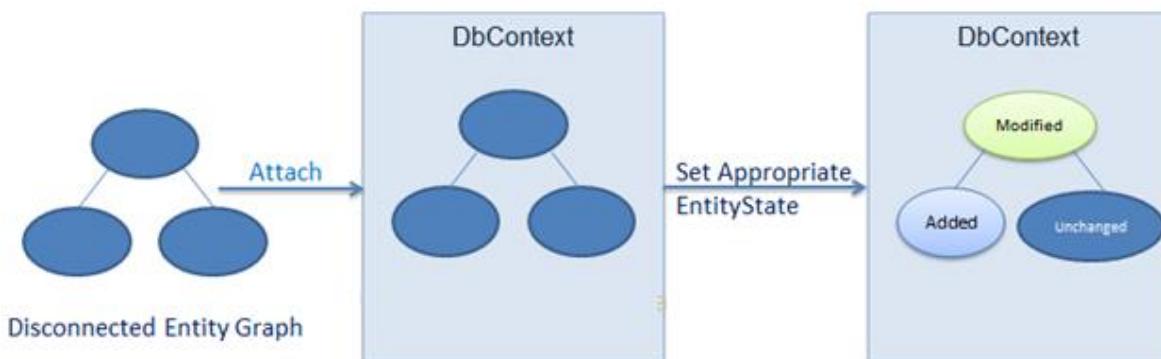
```
using (var ctx = new SchoolDBEntities())
{
    int noOfRowUpdated = ctx.Database.ExecuteSqlCommand("Update student set studentname ='changed student by command' where studentid=1");

    int noOfRowInserted = ctx.Database.ExecuteSqlCommand("insert into student(studentname) values('New Student')");

    int noOfRowDeleted = ctx.Database.ExecuteSqlCommand("delete from student where studentid=1");
}
```

5.7.9. Lưu trữ dữ liệu trong tình huống không kết nối

Việc lưu trữ một thực thể trong tình huống không kết nối tương đối khác với tình huống có kết nối. Có hai việc cần làm khi thao tác với các tập thực thể không kết nối hoặc thậm chí là chỉ một thực thể. Đầu tiên, chúng ta cần đưa các thực thể vào đối tượng context và làm cho đối tượng context theo dõi các thực thể đó. Thứ hai, ta phải thiết lập giá trị EntityState phù hợp cho từng thực thể bởi vì đối tượng context không biết gì về các thao tác đã được thực hiện trên các thực thể không kết nối (vì thế nó không thể tự gán giá trị EntityState được). Hình 5.30 minh họa hai bước vừa đề cập.



Hình 5.30. Hai bước lưu trữ tập thực thể trong tình huống có kết nối

EF cung cấp ba phương thức để gắn các thực thể không kết nối vào một đối tượng context và gán giá trị EntityState cho mỗi thực thể, bao gồm: DbContext.Entry(), DbSet.Add, DbSet.Attach().

Phương thức Entry() của lớp DbContext trả về một thê hiện của lớp DbEntityEntry tương ứng với thực thê được truyền vào. Đối tượng DbEntityEntry cung cấp nhiều thông tin hữu ích về thực thê và có khả năng thực hiện một thao tác trên thực

thể đó. Trong đó, quan trọng nhất là ta có thể thay đổi giá trị của thuộc tính State để chỉ ra trạng thái của thực thể.

```
context.Entry(entity).state = EntityState.Added/Modified/Deleted
```

Phương thức Entry() gắn toàn bộ nhóm thực thể vào một đối tượng context kèm với trạng thái được chỉ định cho thực thể cha và gán trạng thái EntityState cho các thực thể con.

```
var student = new Student() { //Root entity (empty key)
    StudentName = "Bill",
    StandardId = 1,
    Standard = new Standard() //Child entity (with key value)
    {
        StandardId = 1,
        StandardName = "Grade 1"
    },
    Courses = new List<Course>()
    {
        new Course(){ CourseName = "Machine Language" }, //Child entity (empty key)
        new Course(){ CourseId = 2 } //Child entity (with key value)
    }
};

using (var context = new SchoolDBEntities())
{
    context.Entry(student).State = EntityState.Added;

    foreach (var entity in context.ChangeTracker.Entries()){
        Console.WriteLine("{0}: {1}", entity.Entity.GetType().Name, entity.State);
    }
}
```

Kết quả:

```
Student: Added
Standard: Added
Course: Added
Course: Added
```

Trong ví dụ này, nhóm thực thể Student bao gồm cả các thực thể Standard và Course. Câu lệnh context.Entry(student).State = EntityState.Added; gán trạng thái Added cho thực thể cha student và tất cả các thực thể con cho dù các thực thể con chưa giá trị khóa chính là NULL hoặc khác NULL. Vì vậy, cần đặc biệt cẩn thận khi sử dụng phương thức này. Bảng 5.10 mô tả một số hành vi của phương thức Entry tương ứng với trạng thái của thực thể cha.

Bảng 5.10. Danh sách hành vi của phương thức Entry tương ứng với trạng thái của thực thể cha

Trạng thái thực thể cha	Trạng thái các thực thể con
Added	Added
Modified	Unchanged
Deleted	All child entities will be null

Phương thức DbSet.Add() gắn toàn bộ nhóm thực thể vào một đối tượng context và tự động gán trạng thái cho tất cả các thực thể là Added. Việc gọi hàm context.SaveChanges() sẽ thực thi các lệnh INSERT cho mọi thực thể và các mẫu tin sẽ được thêm vào các bảng tương ứng trong cơ sở dữ liệu.

```
//disconnected entity graph
Student disconnectedStudent = new Student() { StudentName = "New Student" };
disconnectedStudent.StudentAddress = new StudentAddress() { Address1 = "Address", City = "City1" };

using (var context = new SchoolDBEntities())
{
    context.Students.Add(disconnectedStudent);
    // get DbEntityEntry instance to check the EntityState of specified entity
    var studentEntry = context.Entry(disconnectedStudent);
    var addressEntry = context.Entry(disconnectedStudent.StudentAddress);

    Console.WriteLine("Student: {0}", studentEntry.State);
    Console.WriteLine("StudentAddress: {0}", addressEntry.State);
}
```

Kết quả:

```
Student: Added
StudentAddress: Added
```

Phương thức DbSet.Attach() gắn toàn bộ nhóm thực thể vào một đối tượng context mới và gán giá trị cho thuộc tính State của tất cả các thực thể là Unchanged.

```
//disconnected entity graph
Student disconnectedStudent = new Student() { StudentName = "New Student" };
disconnectedStudent.StudentAddress = new StudentAddress() { Address1 = "Address", City = "City1" };

using (var context = new SchoolDBEntities())
{
    context.Students.Attach(disconnectedStudent);
    // get DbEntityEntry instance to check the EntityState of specified entity
    var studentEntry = context.Entry(disconnectedStudent);
```

```
var addEntry = context.Entry(disconnectedStudent.StudentAddress);
Console.WriteLine("Student: {0}", studentEntry.State);
Console.WriteLine("StudentAddress: {0}", addEntry.State);
}
```

Kết quả:

```
Student: Unchanged
StudentAddress: Unchanged
```

Việc xóa một thực thể không kết nối cũng tương đối đơn giản. Ta chỉ cần thiết lập trạng thái Deleted cho thực thể đó bằng cách dùng phương thức Entry() rồi gọi phương thức context.SaveChanges().

```
// disconnected entity to be deleted
var student = new Student(){ StudentId = 1 };
using (var context = new SchoolDBEntities())
{
    context.Entry(student).State=System.Data.Entity.EntityState.Deleted;
    context.SaveChanges();
}
```

Trong ví dụ trên, ta có một thể hiện của thực thể Student chỉ chứa thuộc tính khóa chính StudentId. Để xóa một thực thể, đối tượng context chỉ cần yêu cầu thuộc tính khóa chính. Dòng lệnh `context.Entry(student).State = System.Data.Entity.EntityState.Deleted` gắn thực thể student vào đối tượng context và thiết lập trạng thái cho nó là deleted. Khi đó, lệnh SQL DELETE dưới đây sẽ được thực thi trong cơ sở dữ liệu để xóa sinh viên có mã là 1.

```
delete [dbo].[Student]
where ([StudentId] = @0)',N'@0 int',@0=1
```

5.8. Kết chương

Trong chương này, người đọc đã được học cách sử dụng một số thư viện hàm để thao tác với nhiều loại định dạng tập tin khác nhau. .NET Framework cung cấp sẵn các lớp tiện ích để có thể đọc, ghi dữ liệu từ tập tin dạng văn bản lẫn nhị phân. Tuy nhiên, đối với một số định dạng như CSV, XML, các bảng tính Excel, do đặc thù về định dạng, việc sử dụng các lớp có sẵn của .NET khiến lập trình viên phải viết nhiều mã nguồn hơn và dễ gây ra lỗi. Ngoài ra, việc đọc và phân tích các định dạng tập tin như vậy đòi hỏi lập trình viên phải hiểu rõ cấu trúc của tập tin. Do đó, cộng đồng người dùng .NET đã phát triển nhiều gói thư viện để giúp lập trình viên giám bớt công sức và dễ dàng hơn trong việc xử lý các loại tập tin các nhau. Người đọc đã được làm quen với các thư viện CsvHelper để thao tác với tập tin CSV, dùng thư viện EPPlus để xử lý bảng tính Excel. Các gói thư viện này được phân phối thông qua Nuget. Chương này cũng giới thiệu cách cài đặt các thư viện hàm sử dụng Package Manager Console và cửa sổ NuGet Package Manager. Phần cuối chương đã giới thiệu thư viện hàm mới – Entity Framework - được

sử dụng rất phổ biến hiện nay trong việc thao tác với các cơ sở dữ liệu quan hệ. Thông qua đó, người đọc đã học được cách định nghĩa các lớp thực thể, xây dựng lớp ngữ cảnh, truy vấn dữ liệu theo nhiều phương thức khác nhau và cập nhật dữ liệu trong hai tình huống: có kết nối và không có kết nối.

Bài tập:

- Cho tập tin văn bản students.csv có nội dung như dưới đây. Dòng đầu tiên là dòng tiêu đề. Các dòng tiếp theo, mỗi dòng chứa thông tin của một sinh viên. Các trường được phân cách nhau bởi một dấu TAB. Hãy sử dụng các lớp được cung cấp sẵn trong .NET Framework để đọc nội dung tập tin trên, phân tích các trường và tạo một mảng các đối tượng Student.

MSSV	Student Name	Date of Birth	Gender	Grade
00001	John Doe	04/10/1990	Male	5.0
00002	Peter Cloud	12/25/1991	Male	4.5
00003	Mary Currie	08/14/1990	Female	4.6
00004	Sandy Brown	01/09/1993	Female	3.7

- Sử dụng thư viện CsvHelper để đọc tập tin students.csv ở trên và tạo mảng các đối tượng Student.
- Viết chương trình ghi danh sách sinh viên ở câu 2 vào tập tin nhị phân students.bin. Sau đó, đọc nội dung tập tin vừa tạo để hiển thị danh sách sinh viên ban đầu lên màn hình.
- Cho tập tin XML có nội dung như dưới đây. Hãy sử dụng các lớp XmlDocument, XmlTextReader, XmlReader để đọc và phân tích nội dung của tập tin, chuyển thành danh sách các đối tượng Book.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Books.xml stores information about Mahesh Chand and related books --&gt;
&lt;catalog&gt;
    &lt;book ISBN="9831123212" yearpublished="2002"&gt;
        &lt;title&gt;A Programmer's Guide to ADO .NET using C#&lt;/title&gt;
        &lt;author&gt;
            &lt;first-name&gt;Mahesh&lt;/first-name&gt;
            &lt;last-name&gt;Chand&lt;/last-name&gt;
        &lt;/author&gt;
        &lt;publisher&gt;Apress&lt;/publisher&gt;
        &lt;price&gt;44.99&lt;/price&gt;
    &lt;/book&gt;
    &lt;book ISBN="9781484234" yearpublished="2019"&gt;
        &lt;title&gt;Pro Entity Framework Core 2&lt;/title&gt;
        &lt;author&gt;
            &lt;first-name&gt;Adam&lt;/first-name&gt;
            &lt;last-name&gt;Freeman&lt;/last-name&gt;
        &lt;/author&gt;
        &lt;publisher&gt;Apress&lt;/publisher&gt;
        &lt;price&gt;45.09&lt;/price&gt;
    &lt;/book&gt;
&lt;/catalog&gt;</pre>
```

5. Viết chương trình sử dụng LINQ to XML để ghi danh sách đối tượng Book đọc được từ câu 3 vào một tập tin XML mới có tên “publish.xml”.
6. Viết chương trình đọc tập tin “product.xls” chứa thông tin về một danh sách sản phẩm bằng cách sử dụng thư viện hàm ExcelDataReader.
7. Viết chương trình đọc tập tin “product.xls” chứa thông tin về một danh sách sản phẩm bằng cách sử dụng OleDbConnection, OleDbCommand, ...
8. Cài đặt lớp Logger để ghi lại tất cả các lỗi xảy ra trong lúc chạy chương trình vào một tập tin dạng văn bản có tên là “error.log”. Với mỗi lỗi, chương trình cần lưu vào tập tin 2 dòng. Dòng thứ nhất chứa ngày giờ xảy ra lỗi, tên lớp chứa mã nguồn gây ra lỗi, nội dung thông báo lỗi. Dòng thứ hai chứa thông tin Stack Trace (thông tin chi tiết về quá trình gọi hàm) để cho biết chi tiết về lỗi xảy ra.
9. Viết chương trình để nạp (import) dữ liệu từ tập tin XML trong câu 4 vào cơ sở dữ liệu bằng cách dùng LINQ to XML và Entity Framework.
10. Tìm hiểu về cách thực thi các thủ tục (Stored Procedure) với Entity Framework và viết thành một báo cáo (20-30 trang). Mỗi sinh viên chọn một chủ đề, thiết kế cơ sở dữ liệu và viết code để minh họa.

CHƯƠNG 6. KIỂM SOÁT GIAO DỊCH NÂNG CAO

Trong quá trình thực thi các lệnh về cơ sở dữ liệu, có nhiều khi cần thực hiện đồng thời nhiều lệnh mà có cùng một mục đích. Một ví dụ điển hình là việc chuyển tiền từ một tài khoản này sang một tài khoản khác. Giao dịch này gồm hai lệnh cập nhật (Update), một lệnh dùng để rút tiền từ tài khoản này và một lệnh để chuyển số tiền vừa rút sang tài khoản khác. Cả hai lệnh cập nhật này được xem là một *giao dịch* duy nhất vì cả hai đều phải được xác nhận hoặc chúng phải được hủy bỏ nếu một trong các thao tác thất bại. Nếu không, tiền trong tài khoản có thể sẽ bị mất.

Khi các dòng lệnh truy vấn cơ sở dữ liệu được gom lại với nhau nhằm thực hiện một mục đích nào đó, toàn bộ quá trình thực thi đó người ta gọi là một giao dịch (*transaction*). Nếu được thực thi, hoặc tất cả các thay đổi tạo ra bởi giao dịch phải được xác nhận hoặc tất cả phải được hủy bỏ. Về mặt logic, các lệnh riêng rẽ trong SQL được xem là một đơn vị công việc.

Các cơ sở dữ liệu hiện nay đều có khả năng quản lý nhiều người dùng và nhiều chương trình truy xuất đến cơ sở dữ liệu cùng lúc. Mỗi chương trình thực thi nhiều giao dịch trên cùng một cơ sở dữ liệu. Điều này còn được gọi là các giao dịch đồng thời (*concurrent transaction*) vì chúng được thi tại cùng một thời điểm. Phần mềm quản trị cơ sở dữ liệu phải có khả năng đáp ứng các yêu cầu về các giao dịch đồng thời này cũng như duy trì được tính toàn vẹn của dữ liệu được lưu trong các bảng. Bạn có thể kiểm soát các mức độ độc lập (*the level of isolation*) tồn tại giữa các giao dịch của mình với các giao dịch khác đang thực thi trong cơ sở dữ liệu.

Chương này đi sâu vào việc phân tích và kiểm soát các giao dịch nâng cao sử dụng SQL Server và ADO .Net. Những nội dung chính sẽ được trình bày:

- Lớp SqlTransaction;
- Thiết lập điểm lưu (Save point);
- Gán mức độ độc lập cho giao dịch cơ sở dữ liệu;
- Các chế độ khóa của SQL Server;
- Chặn giao dịch.

6.1. Các lớp SqlTransaction

Transaction là một đối tượng được dùng để biểu diễn một giao dịch trong cơ sở dữ liệu. Có ba lớp dùng để xử lý giao dịch bao gồm OleDbTransaction, OdbcTransaction và SqlTransaction. Để biểu diễn và xử lý cơ sở dữ liệu với SQL Server, ta dùng lớp SqlTransaction, lớp này được kế thừa từ lớp System.Data.Common.DbTransaction.

Các phương thức và thuộc tính của lớp SqlTransaction được nêu chi tiết trong Bảng 6.1 (Nguyễn, Nguyễn, 2018). Chương trình 6.1 biểu diễn một ví dụ về Transaction khi thực thi hai lệnh thêm dữ liệu vào bảng Region, nếu quá trình thêm xảy ra lỗi, chương trình sẽ hủy bỏ cả hai lệnh đã thực hiện bằng lệnh Rollback.

Bảng 6.1. Các thuộc tính và phương thức của lớp SqlTransaction

Thuộc tính/ Phương thức	Kiểu	Mô tả
Connection	SqlConnection	Lấy kết nối cho giao dịch cơ sở dữ liệu
IsolationLevel	IsolationLevel	Lấy mức độ độc lập của giao dịch.
Commit()	void	Xác nhận các lệnh SQL trong giao dịch.
Rollback()	void	Hủy bỏ các lệnh SQL trong giao dịch. Phương thức này có nhiều tham số truyền vào.
Save()		Tạo một điểm lưu (savepoint) trong giao dịch để có thể hủy bỏ một phần của giao dịch. Phương thức này nhận một tham số kiểu chuỗi làm tên của savepoint, sau đó, có thể quay lui (roll back) giao dịch về savepoint.

Chương trình 6.1. Ví dụ về Transaction trong C#

```
private static void ExecuteSqlTransaction(string connStr)
{
    using (SqlConnection connection = new SqlConnection(connStr))
    {
        // Mở kết nối, tạo lệnh giao dịch
        connection.Open();
        SqlCommand command = connection.CreateCommand();
        // Khai báo một giao dịch có tên SampleTransaction
        SqlTransaction transaction;
        transaction = connection.BeginTransaction("SampleTransaction");
        command.Connection = connection;
        command.Transaction = transaction;
        try
        {
            // Tạo 2 lệnh giao dịch trên bảng Account
            command.CommandText = "insert into Account (AccountName,
Password, FullName, Email, Tell) VALUES (quytd1, '12345', 'Thái Duy Quý
01', 'thaiduyquy@gmail.com', '0963555666')";
            command.ExecuteNonQuery();
            command.CommandText = "insert into Account (AccountName,
Password, FullName, Email, Tell) VALUES (quytd2, '123@456', 'Thái Duy Quý
02', 'minhnhat111@gmail.com', '0963010140')";
            command.ExecuteNonQuery();
            // Thực thi lệnh giao dịch
            transaction.Commit();
        }
        catch (Exception ex)
        {
            // Trong trường hợp xảy ra lỗi khi thực hiện giao dịch, thì
            // hủy bỏ các dòng lệnh đã thực hiện
            transaction.Rollback();
        }
    }
}
```

6.2. Thiết lập điểm lưu (Save point)

Trong quá trình thực hiện giao dịch, ta có thể tạo ra một điểm lưu ở một số vị trí nhất định. Điều này cho phép hủy bỏ các thay đổi tạo ra trên dữ liệu sau một điểm lưu nào đó bằng cách sử dụng lệnh Rollback để quay ngược lại. Công việc thiết lập điểm lưu thực sự hữu ích khi chúng ta có một giao dịch bao gồm nhiều lệnh, nếu có một lỗi sau mỗi điểm lưu đã thiết lập, chúng ta không cần hủy bỏ toàn bộ giao dịch và làm lại từ đầu.

6.2.1. Thiết lập điểm lưu bằng lệnh trong SQL Server

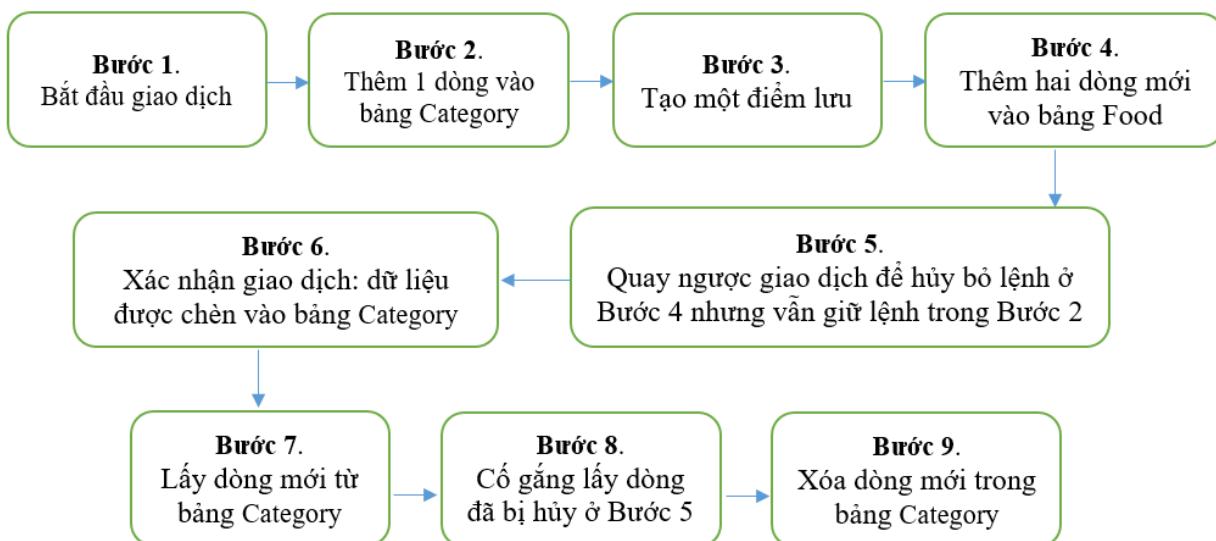
Trong SQL Server, có thể thiết lập điểm lưu bằng lệnh SAVE TRANSACTION hoặc có thể viết tắt là SAVE TRANS. Cú pháp lệnh như sau:

```
SAVE TRANSACTION {savepointName| @savepointVar}
```

Trong đó:

- *savepointName*: là một chuỗi chứa tên gán cho điểm lưu.
- *@savepointVar*: là một biến của T-SQL chứa tên của điểm lưu. Biến này có kiểu char, varchar, nchar hoặc nvarchar.

Chương trình 6.2 thực hiện 9 bước (Hình 6.1) để thiết lập điểm lưu trong SQL Server. Bắt đầu từ việc thêm dữ liệu cho bảng Category, Food đến khi kết thúc chương trình là việc cố gắng lấy ra dữ liệu mặc dù đã hủy bỏ thao tác trước đó.



Hình 6.1. Các bước thực hiện thiết lập điểm lưu

Chương trình 6.2. Tạo điểm lưu bằng các lệnh trong SQL Server

```
/*
    Chương trình minh họa tạo điểm lưu
    Database: RM (RestaurantManagement)
    Tables: Category, Food
*/
Use QLSV
--Bước 1. Bắt đầu giao dịch
BEGIN TRANSACTION

--Bước 2. Chèn 1 dòng vào bảng Category
insert into Category([Name], [Type]) values (N'Điểm tâm', 1)

--Bước 3. Thiết lập điểm lưu
SAVE TRANSACTION SavePoint1

--Bước 4. Chèn 2 dòng vào bảng Food, phải lấy ID của Category trước
insert into Food([Name],[Unit], FoodCategoryID, Price, Notes)
    values(N'Miến xào', N'Dĩa', 20, 35000, N'Dành cho 3 người')
insert into Food([Name],[Unit], FoodCategoryID, Price, Notes)
    values(N'Mì xào', N'Dĩa', 20, 30000, N'Dành cho 2 người')

--Bước 5. Quay ngược giao dịch để hủy bỏ 2 lệnh ở Bước 4 nhưng vẫn giữ
lệnh tại Bước 2
ROLLBACK TRANSACTION SavePoint1

--Bước 6. Xác nhận giao dịch: dữ liệu được chèn vào bảng Category
COMMIT TRANSACTION

--Bước 7. Lấy dòng mới từ bảng Category
select * from Category where [Name]= 'Điểm tâm'

--Bước 8. Thử lấy dòng bị hủy tại Bước 5
select * from Food where [Name]= 'Miến xào'

--Bước 9. Xóa dòng mới chèn trong bảng Category
delete from Category where [Name]= 'Điểm tâm'
```

6.2.2. Tạo điểm lưu bằng đối tượng SqlTransaction

Có thể dùng đối tượng SqlTransaction để tạo điểm lưu bằng cách gọi phương thức Save và truyền vào tham số là tên của điểm lưu. Sau đó, để hủy bỏ giao dịch tại điểm lưu vừa tạo, có thể dùng phương thức Rollback.

Ví dụ: Giả sử ta tạo ra một đối tượng của lớp SqlTransaction như sau:

```
SqlTransaction sqlTransaction = Conn.BeginTransaction();
```

Khi đó, đối tượng được gán cho thuộc tính của SqlCommand như sau:

```
SqlCommand sqlCommand = Conn.CreateCommand();
sqlCommand.Transaction = sqlTransaction;
```

Chương trình 6.3 minh họa quá trình thiết lập điểm lưu trong giao dịch sử dụng đối tượng SqlTransaction để thực hiện. Chương trình này thực hiện lại Chương trình 6.2 bằng ngôn ngữ C# trên ADO .Net. Sau khi khai báo và mở đối tượng kết nối SqlConnection, mã lệnh của chương trình cũng bao gồm 9 bước như sau:

- Bước 1: Tạo đối tượng SqlTransaction, tạo đối tượng SqlCommand và gán thuộc tính Transaction cho đối tượng vừa tạo;
- Bước 2: Chèn một dòng vào bảng Category;
- Bước 3: Thiết lập điểm lưu với tên “SavePoint1”;
- Bước 4: Chèn 2 dòng dữ liệu vào bảng Food;
- Bước 5: Thực thi lệnh Rollback để trường hợp xảy ra lỗi sẽ bỏ qua Bước 4, tới Bước 6.
- Bước 6: Xác nhận, dữ liệu được chèn vào bảng Category (Bước 3);
- Bước 7: Lấy dữ liệu từ bảng Category;
- Bước 8: Lấy dữ liệu từ bảng Food;
- Bước 9: Xóa dữ liệu từ bảng Category.

Chương trình 6.3. Tạo điểm lưu dùng đối tượng SqlTransaction

```
/// <summary>
/// Phương thức thực thi một số lệnh Savepoint
/// </summary>
/// <param name="connStr">Chuỗi kết nối</param>
public static void ExecuteSavepoint(string connStr)
{
    using (SqlConnection Conn = new SqlConnection(connStr))
    {
        // Mở kết nối, tạo lệnh giao dịch
        Conn.Open();
        //Bước 1. Tạo đối tượng SqlTransaction
        // Tạo đối tượng SqlCommand và gán thuộc tính Transaction cho
đối tượng vừa tạo
        SqlTransaction sqlTransaction = Conn.BeginTransaction();
        SqlCommand sqlCommand = Conn.CreateCommand();
        sqlCommand.Transaction = sqlTransaction;
        // Bước 2. Chèn 1 dòng vào bảng Category
        string Sql = "insert into Category([Name], [Type]) values
(N'Điểm tâm', 1)";
        sqlCommand.CommandText = Sql;
        sqlCommand.ExecuteNonQuery();

        // Bước 3. Thiết lập điểm lưu với tên "SavePoint1"
        sqlTransaction.Save("SavePoint1");
        try
        {
            // Bước 4. Chèn 2 dòng vào bảng Food, ID của Category ở trên
```

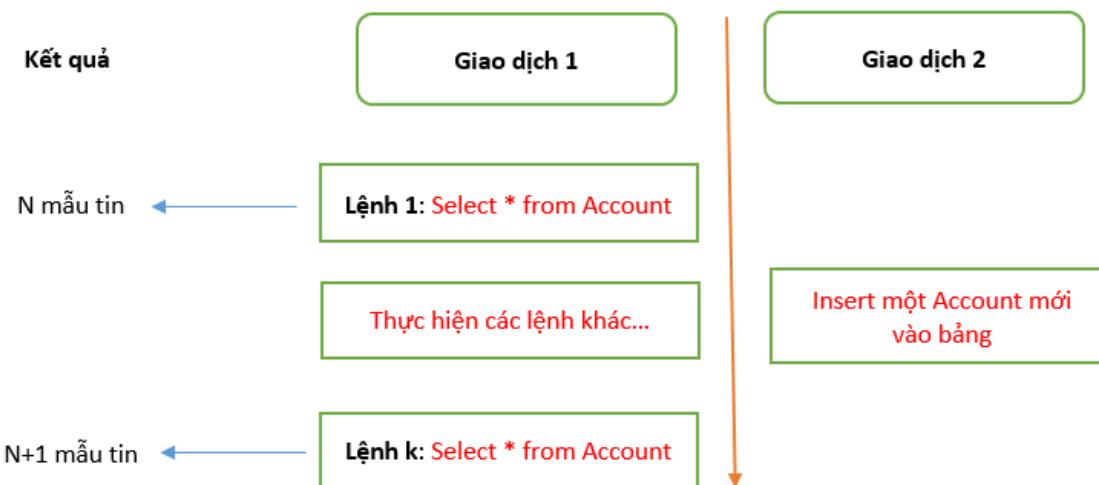
```
        string Sql1 = "insert into Food values(N'Cơm chiên Dương  
Châu',N'Đĩa',15,50000,N'Dành cho 5 người')";  
        sqlCommand.CommandText = Sql1;  
        sqlCommand.ExecuteNonQuery();  
        string Sql2 = "Insert into Food values(N'Cơm chiên cá mặn  
,N'Đĩa',15,45000,N'Dành cho 5 người')";  
        sqlCommand.CommandText = Sql2;  
        sqlCommand.ExecuteNonQuery();  
    }  
    catch  
    {  
        // Bước 5. Nếu xảy ra lỗi sẽ bỏ qua Bước 4, tới Bước 6.  
        sqlTransaction.Rollback("SavePoint1");  
    }  
    //Bước 6. Xác nhận giao dịch: dữ liệu được chèn vào bảng  
Category  
    sqlTransaction.Commit();  
    //Bước 7. Lấy dữ liệu từ bảng Category  
    string Sql3 = "Select * from Category";  
    sqlCommand.CommandText = Sql3;  
    SqlDataReader dataReader = sqlCommand.ExecuteReader();  
    while (dataReader.Read())  
    {  
        String Str1 = String.Format("Loại thực ăn: {0}, Kieu:  
{1}", dataReader["Name"], dataReader["Type"]);  
        Console.WriteLine(Str1);  
    }  
    dataReader.Close();  
  
    //Bước 8. Lấy dữ liệu từ bảng Food  
    string Sql4 = "Select * from Food";  
    sqlCommand.CommandText = Sql4;  
    SqlDataReader dataReader1 = sqlCommand.ExecuteReader();  
    while (dataReader1.Read())  
    {  
        String Str2 = String.Format("Tên món ăn: {0}, Gia: {1},  
DVT: {2}, Ghi chú: {3}", dataReader1["Name"], dataReader1["Price"],  
dataReader1["Unit"], dataReader1["Notes"]);  
        Console.WriteLine(Str2);  
    }  
    dataReader1.Close();  
  
    //Bước 9. Xóa dữ liệu từ bảng Category  
    string Sql5 = "delete from Category where ID =20";  
    sqlCommand.CommandText = Sql5;  
    int count = sqlCommand.ExecuteNonQuery();  
    String Str = String.Format("Số mục tin đã xóa: {0}", count);  
    Console.WriteLine(Str);  
}  
}
```

6.3. Mức độ độc lập giao dịch cơ sở dữ liệu

Mức độ độc lập của giao dịch là mức độ tách biệt của các thay đổi được tạo ra bởi một giao dịch này với các giao dịch đồng thời được thực hiện bởi những người dùng hay ứng dụng khác. Trước khi đi vào việc gán mức độ độc lập cho các giao dịch đồng thời, ta cần phải hiểu các vấn đề có thể xảy ra khi các giao dịch cùng lúc tác động lên cùng một số dòng trong bảng.

6.3.1. Phantoms

Phantoms xảy ra khi một giao dịch thực thi một lệnh truy vấn hai lần nhưng lại trả về kết quả khác nhau số dòng. Điều này là do khi đang diễn ra một giao dịch thì một giao dịch khác chèn vào một mẫu tin với cùng điều kiện. Mô hình trong Hình 6.2 cho thấy có hai giao dịch đồng thời xảy ra, trong khi giao dịch 1 đang thực hiện công việc thì giao dịch 2 đã chèn thêm một mẫu tin mới, do vậy, khi thực hiện lệnh select cuối cùng của giao dịch 1 thì có 1 dòng mới được thêm vào bảng, dòng mới thêm vào được gọi là “phantom” (ảo).



Hình 6.2. Giao dịch xảy ra trường hợp *Phantom*

Chương trình 6.4 sử dụng SQL Server minh họa quá trình xảy ra *Phantom* của Hình 6.2 khi cho 2 giao dịch chạy đồng thời tại cùng thời điểm. Để thực hiện ví dụ này, cần phải mở hai cửa sổ, chạy giao dịch 1 trước, sau đó chạy giao dịch 2. Kết quả khi chạy xong giao dịch 1 được minh họa trong Hình 6.3 cho thấy có 1 dòng đã được thêm vào khi thực hiện xong giao dịch.

Chương trình 6.4. Ví dụ về *Phantom* khi thực hiện hai giao dịch đồng thời

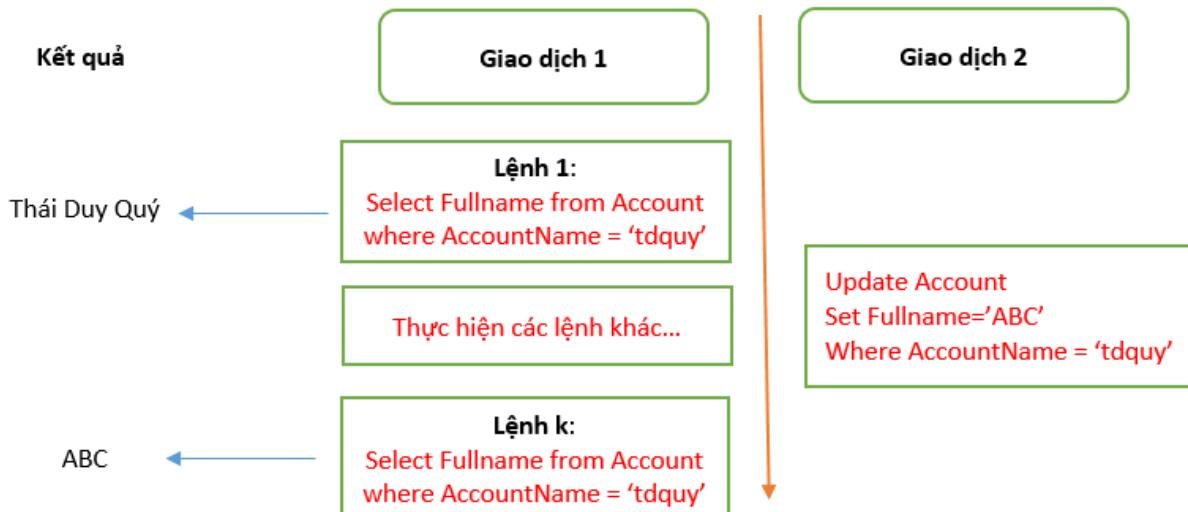
<pre>--Giao dịch 1 BEGIN TRANSACTION -- Truy vấn dữ liệu Lần 1 select AccountName, FullName from Account -- Chờ 10 giây để chạy Giao dịch 2 waitfor delay '00:00:10'</pre>	<pre>--Giao dịch 2 BEGIN TRANSACTION -- Chèn dữ liệu insert into Account values('tdquy1', '', 'Thái Duy Quý 1', '', null) COMMIT TRANSACTION</pre>
--	--

```
-- Truy vấn dữ liệu lần 2
select AccountName, FullName
from Account
COMMIT TRANSACTION
```

Hình 6.3. Kết quả của Giao dịch 1 khi chạy đồng thời 2 giao dịch

6.3.2. Nonrepeatable reads

Nonrepeatable reads xảy ra khi một giao dịch thực hiện 2 lần cùng một truy vấn nhưng lại trả về kết quả khác nhau. Điều này xảy ra khi vừa truy vấn xong lần 1 thì một giao dịch khác cập nhật mẫu tin với cùng điều kiện. Mô hình trong Hình 6.4 cho thấy có hai giao dịch đồng thời xảy ra, trong khi giao dịch 1 đang thực hiện công việc thì giao dịch 2 đã cập nhật mẫu tin với cùng điều kiện, do vậy, khi thực hiện lệnh select cuối cùng thì có sự sai khác so với lệnh trước đó.



Hình 6.4. Giao dịch xảy ra trường hợp *Nonrepeatable reads*

Chương trình 6.5 sử dụng SQL Server minh họa quá trình xảy ra *Nonrepeatable reads* của Hình 6.4 khi cho 2 giao dịch chạy đồng thời tại cùng thời điểm. Tương tự như trên, để thực hiện ví dụ này, cần phải mở hai cửa sổ, chạy giao dịch 1 trước, sau đó chạy giao dịch 2. Kết quả khi chạy xong giao dịch 1 được minh họa trong Hình 6.5, kết quả cho thấy có sự khác nhau về dữ liệu khi thực hiện xong giao dịch với cùng một lệnh truy vấn.

Chương trình 6.5. Ví dụ về Nonrepeatable reads khi thực hiện hai giao dịch đồng thời

```
--Giao dịch 1
BEGIN TRANSACTION
    -- Truy vấn dữ liệu Lần 1
    select FullName from Account
    where AccountName='tdquy'
    -- Chờ 10 giây để chạy Giao
    dịch 2
    waitfor delay '00:00:10'

    -- Truy vấn dữ liệu lần 2
    select FullName from Account
    where AccountName='tdquy'
COMMIT TRANSACTION

--Giao dịch 2
BEGIN TRANSACTION
    -- Sửa dữ liệu cùng điều kiện
    update Account
    set FullName = 'ABC'
    where AccountName='tdquy'
COMMIT TRANSACTION
```

FullName
Thái Duy Quý

FullName
ABC

Hình 6.5. Kết quả của Giao dịch 1 khi chạy đồng thời 2 giao dịch

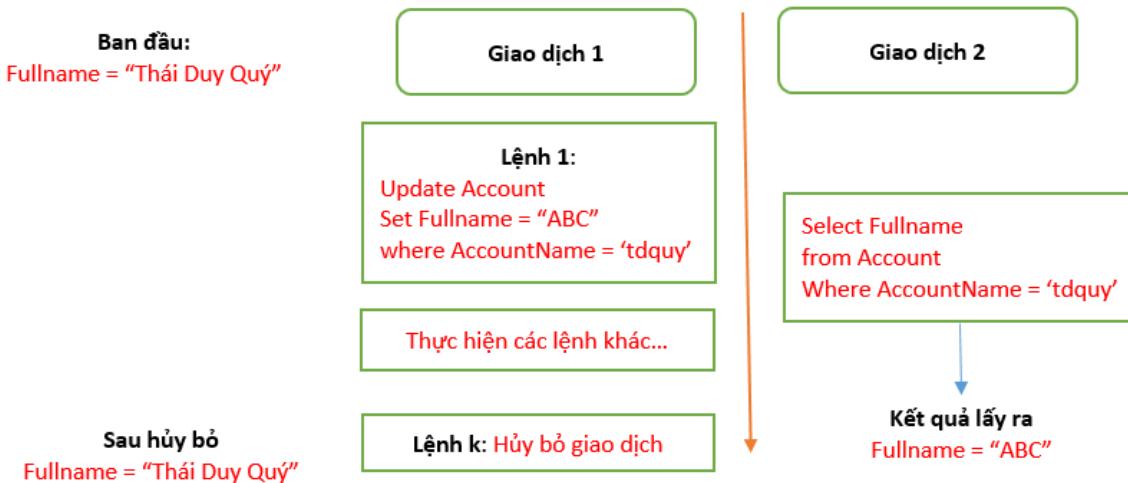
6.3.3. Dirty reads

Dirty reads xảy ra khi một giao dịch thực hiện việc cập nhật nhưng sau đó lại hủy bỏ lệnh đó, khi vừa thực hiện cập nhật thì có một giao dịch khác lại truy vấn để lấy dữ liệu. Như vậy, dữ liệu vừa lấy được thực ra là dữ liệu rác vì người dùng đã hủy bỏ lệnh. Mô hình trong Hình 6.6 minh họa quá trình này, khi giao dịch 1 đang thực hiện cập nhật mẫu tin thì giao dịch 2 đã lấy ra thông tin mẫu tin, tuy nhiên giao dịch 1 lại bị hủy bỏ, do vậy, thông tin mà giao dịch 2 lấy ra thực ra là thông tin rác.

Chương trình 6.6 sử dụng SQL Server minh họa quá trình xảy ra *Dirty read* được mô tả như trong Hình 6.6. Vì SQL Server mặc định không cho phép lấy dữ liệu dạng này nên phải thêm dòng sau trên đầu của Giao dịch 2:

```
Set transaction isolation level read uncommitted
```

Kết quả chạy của giao dịch 2 (Hình 6.7) cho thấy kết quả đó là kết quả rác, vì giao dịch 1 đã hủy bỏ lệnh nhưng giao dịch 2 vẫn lấy được dữ liệu.



Hình 6.6. Giao dịch xảy ra trường hợp *Dirty reads*

Chương trình 6.6. Ví dụ về *Dirty reads* khi thực hiện hai giao dịch đồng thời

```

--Giao dịch 1
BEGIN TRANSACTION
-- Cập nhật dữ liệu
update Account
set FullName = 'ABC'
where AccountName= 'tdquy'
-- Chờ 10 giây để chạy Giao dịch 2
waitfor delay '00:00:10'
-- Hủy bỏ lệnh cập nhật
ROLLBACK TRANSACTION

-- Cho phép đọc dữ liệu rác
Set transaction isolation level read
uncommitted
--Giao dịch 2
BEGIN TRANSACTION
-- Lấy dữ liệu
Select FullName from Account
where AccountName = 'tdquy'
COMMIT TRANSACTION
    
```

Results	
	FullName
1	ABC

Hình 6.7. Kết quả của Giao dịch 2 khi đang chạy đồng thời 2 giao dịch

6.3.4. Thiết lập mức độ độc lập giao dịch trong SQL Server

Để giải quyết các vấn đề trên đây, các hệ quản trị cơ sở dữ liệu sử dụng và thực thi nhiều mức độ độc lập giao dịch để ngăn chặn các giao dịch lấy về dữ liệu không có ý nghĩa. Chuẩn SQL định nghĩa bốn mức độ độc lập như được liệt kê trong Bảng 6.2.

Mặc định của SQL Server là *Read Committed* và được chấp nhận với hầu hết các ứng dụng. Khi đặt mức độ độc lập của giao dịch là *Serializable* thì bất kỳ dòng nào được truy xuất trong một giao dịch đến trước đều bị “khóa”, nghĩa là giao dịch khác không

thể thực hiện việc cập nhật cho các dòng đó. Do vậy, chỉ nên sử dụng lệnh *Serializable* khi phải bảo đảm giao dịch đang thực hiện độc lập hoàn toàn với các giao dịch khác.

Bảng 6.2. Các mức độ độc lập trong giao dịch của SQL

Mức độ độc lập	Mô tả
Read Uncommitted	Cho phép thực thi lệnh để lấy về dữ liệu <i>Phantoms</i> , <i>Nonrepeatable reads</i> , và <i>Dirty reads</i> .
Read Committed	Cho phép thực thi lệnh để lấy về dữ liệu <i>Phantoms</i> và dữ liệu <i>Nonrepeatable reads</i> . Lệnh này không cho phép lấy về dữ liệu <i>Dirty reads</i> .
Repeatable Read	Cho phép thực thi lệnh để lấy về dữ liệu <i>Phantoms</i> nhưng không cho phép lấy dữ liệu <i>Nonrepeatable reads</i> và dữ liệu <i>Dirty reads</i> .
Serializable	Không cho phép thực thi lệnh để lấy về các loại dữ liệu trên. Đây là thiết lập mặc định của chuẩn SQL.

Để đặt mức độ độc lập của giao dịch trong SQL Server, ta dùng lệnh *SET TRANSACTION ISOLATION LEVEL*, cú pháp như sau:

```
SET TRANSACTION ISOLATION LEVEL {
    Read committed |
    Read uncommitted |
    Repeatable read |
    Serializable
}
```

Ví dụ: Nếu muốn Chương trình 6.4, 6.5, và 6.6 không cho phép tạo ra các dữ liệu dạng *Phantoms*, *Nonrepeatable reads*, và *Dirty reads*, ta có thể đặt mức độ độc lập của các giao dịch có kiểu là *Serializable*, có thể viết dòng lệnh sau trên đầu mỗi giao dịch:

```
SET TRANSACTION ISOLATION LEVEL Serializable
```

Khi đó, giao dịch đó sẽ bị “khóa”, các giao dịch khác phải chờ đợi cho đến khi giao dịch đó hoàn thành thì mới thực hiện được, do vậy dữ liệu sẽ không bị “rác”.

6.3.5. Gán mức độ độc lập giao dịch trong ADO .Net

Tương tự như SQL, ADO .Net cũng hỗ trợ một số mức độ độc lập giao dịch, thể hiện trong enum *System.Data.IsolationLevel*. Ngoài các mức độ như trong SQL, ADO .Net còn hỗ trợ thêm mức độ độc lập *Chaos* và *Unspecified*, chi tiết như trong Bảng 6.3.

Để thực thi giao dịch độc lập trong ADO .Net, đầu tiên cần tạo một đối tượng *SqlTransaction* bằng phương thức *BeginTransaction()*. Phương thức có thể truyền vào các tham số như sau:

- *myIsolationLevel*: Là mức độ độc lập của giao dịch, tham số này nhận một trong các giá trị từ enum *System.Data.IsolationLevel* như trong Bảng 6.3;
- *transactionName*: Là chuỗi tên của giao dịch.

Bảng 6.3. Các mức độ độc lập trong giao dịch trong ADO .Net

Mức độ độc lập	Mô tả
Chaos	Chờ các thay đổi không thể ghi đè từ các giao dịch độc lập.
Read Uncommitted	Cho phép thực thi lệnh để lấy về dữ liệu <i>Phantoms</i> , <i>Nonrepeatable reads</i> , và <i>Dirty reads</i> .
ReadCommitted	Cho phép thực thi lệnh để lấy về dữ liệu <i>Phantoms</i> và dữ liệu <i>Nonrepeatable reads</i> . Lệnh này không cho phép lấy về dữ liệu <i>Dirty reads</i> .
RepeatableRead	Cho phép thực thi lệnh để lấy về dữ liệu <i>Phantoms</i> nhưng không cho phép lấy dữ liệu <i>Nonrepeatable reads</i> và dữ liệu <i>Dirty reads</i> .
Serializable	Không cho phép tạo ra các loại dữ liệu trên.
Unspecified	Một mức độ độc lập khác mà người dùng chỉ định nhưng không thể xác định.

Ví dụ: Đoạn mã sau đây khai báo đối tượng SqlConnection kết nối tới cơ sở dữ liệu RestaurantManagement của SQL Server. Sau đó tạo đối tượng SqlTransaction tên là myTransaction bằng cách gọi phương thức BeginTransaction(). Phương thức này truyền vào tham số là IsolationLevel.Serializable. Tiếp theo, một đối tượng SqlCommand được tạo ra và thuộc tính Transaction của nó lấy giá trị của đối tượng SqlTransaction ở trên.

```
// Khai báo đối tượng SqlConnection với chuỗi kết nối truyền vào
SqlConnection myConnection = new SqlConnection(StrConnection);
// Khao báo đối tượng SqlTransaction với kiểu Serializable
SqlTransaction myTransaction =
    myConnection.BeginTransaction(IsolationLevel.Serializable);

// Khai báo đối tượng SqlCommand và gán thuộc tính Transaction
SqlCommand sqlCommand = myConnection.CreateCommand();
sqlCommand.Transaction = myTransaction;
```

Khi đó, lệnh thực thi bởi đối tượng SqlCommand sẽ được thực hiện trong một giao dịch có kiểu Serializable. Ví dụ sau thực hiện các lệnh Insert và lệnh Update, dòng cuối cùng là xác nhận các lệnh bằng cách gọi phương thức Commit của đối tượng SqlTransaction.

```
// lệnh Insert
string insertText = "Insert into Category (Name,Type) values (N'Nước
uống',0)";
sqlCommand.CommandText = insertText;
int rows1 = sqlCommand.ExecuteNonQuery();

// Lệnh Update
int id = 16; // Cần cập nhật ID này
string updateText = "Update Category set Name = N'Nước lọc' where ID
= "+ id;
sqlCommand.CommandText = updateText;
int rows2 = sqlCommand.ExecuteNonQuery();
// Xác nhận các lệnh trên
myTransaction.Commit();
```

6.4. Các chế độ khóa của SQL Server

Chế độ khóa (*locks*) của SQL Server được sử dụng để thực thi sự độc lập giao dịch và bảo đảm tính toàn vẹn của thông tin. Các khóa ngăn chặn người dùng đọc dữ liệu hay thay đổi một dòng khi dòng đó đang được cập nhật bởi một người dùng khác.

6.4.1. Các loại khóa

Bảng 6.4 thống kê một số loại khóa thông dụng trong SQL Server. Các khóa được liệt kê tăng dần theo mức độ chi tiết (*locking granularity*) của đối tượng khóa. Mức độ chi tiết ám chỉ kích thước của tài nguyên đang bị khóa. Chẳng hạn, row lock (khóa dòng) tốt hơn page lock (khóa trang).

Bảng 6.4. Các loại khóa thông dụng trong SQL Server

Loại khóa	Mô tả
Row (RID)	Khóa được gán trên dòng của bảng, dùng để xác định tính duy nhất của dòng.
Key (KEY)	Khóa được đặt trên một dòng cùng với chỉ mục, dùng trong các giao dịch kiểu <i>Serializable</i> .
Page (PAG)	Khóa được áp trên một trang.
Extent (EXT)	Khóa áp dụng cho một phạm vi mở rộng hay các chỉ mục liên tiếp nhau.
Table (TAB)	Khóa được thiết lập trên một bảng dùng để khóa các dòng và chỉ mục có trong bảng.
Database (DB)	Khóa toàn bộ cơ sở dữ liệu.

6.4.2. Các chế độ khóa

Ngoài các loại khóa được mô tả như trong Bảng 6.4, SQL Server cũng có nhiều chế độ khóa khác nhau, dùng để xác định bởi mức độ của khóa được đặt trên tài nguyên. Bảng 6.5 liệt kê một số chế độ khóa thông dụng.

Bảng 6.5. Một số chế độ khóa thông dụng trong SQL Server

Chế độ khóa	Mô tả
Shared (S)	Chế độ này chỉ ra rằng một giao dịch chuẩn bị đọc dữ liệu từ một tài nguyên khi dùng lệnh Select ngăn chặn các giao dịch khác thay đổi trên các tài nguyên bị khóa.
Update (U)	Chế độ này chỉ ra rằng một giao dịch muốn cập nhật một tài nguyên dùng lệnh Insert, Update, hay Delete. Khóa phải được tăng cường thành một khóa riêng trước khi giao dịch thực sự xảy ra.
Exclusive (X)	Cho phép các giao dịch cập nhật tài nguyên dùng lệnh Insert, Update, hay Delete. Các giao dịch khác không thể đọc hoặc ghi đè lên tài nguyên đang có khóa riêng.
Intent shared (IS)	Chế độ mở rộng chế độ Shared bằng cách tăng độ mức độ chi tiết hơn trong tài nguyên đang dùng. Ví dụ, khi đặt khóa IS lên một bảng thì có thể ám chỉ rằng giao dịch có thể đặt khóa Shared trên trang (pages) hoặc dòng (rows) trong bảng đó.
Intent exclusive (IX)	Chế độ này chỉ ra rằng giao dịch muốn đặt một khóa riêng lên tài nguyên với mức độ chi tiết hơn. Khi đó các giao dịch khác không thể đặt khóa dành riêng lên tài nguyên đã có khóa IX.

Bảng 6.5. Một số chế độ khóa thông dụng trong SQL Server (tiếp)

Chế độ khóa	Mô tả
Shared with intent exclusive (SIX)	Chế độ này chỉ ra rằng giao dịch muốn đọc tất cả các tài nguyên có mức độ chi tiết hơn và cập nhật một vài tài nguyên nào đó. Ví dụ, đặt một khóa SIX lên một bảng đồng nghĩa với việc giao dịch muốn đọc tất cả các dòng trong bảng và cập nhật một vài dòng trong đó. Các giao dịch khác không thể đặt khóa dành riêng trên một tài nguyên đã có khóa SIX.

6.5. Các phương pháp chặn giao dịch

6.5.1. Chặn giao dịch

Một giao dịch có thể ngăn chặn hoặc không cho một giao dịch khác lấy tài nguyên mà nó đang chiếm giữ bởi khóa. Trong ví dụ tại Chương trình 6.7, giả sử có hai giao dịch cập nhật dữ liệu cùng tác động lên 1 dòng. Giao dịch 1 có thời gian trễ là 5 giây chạy trước, khi đó Giao dịch 2 phải đợi cho đến khi Giao dịch 1 chạy xong thì mới thực hiện được.

Chương trình 6.7. Ví dụ về chặn giao dịch

```
use RestaurantManagement
--Giao dịch 1
BEGIN TRANSACTION
    -- Cập nhật dữ liệu
    update Account
    set FullName = 'ABCD'
    where AccountName='tdquy'
    -- Chờ 10 giây để chạy Giao dịch 2
    waitfor delay '00:00:5'
    -- Thực thi lệnh cập nhật
COMMIT TRANSACTION

use RestaurantManagement
--Giao dịch 2
BEGIN TRANSACTION
    -- Cập nhật dữ liệu
    update Account
    set FullName = 'XYZ'
    where AccountName='tdquy'
    -- Thực thi lệnh cập nhật
    COMMIT TRANSACTION
```

Hình 6.8 cho thấy hai giao dịch đang được thực thi bởi Query Analyzer. Giao dịch 1 được hiển thị trong phần bên trái, và đang chặn Giao dịch 2 ở bên phải.

Hình 6.8. Minh họa quá trình chặn giao dịch

6.5.2. Đặt thời gian hủy bỏ cho khóa

Mặc định, các lệnh trong SQL sẽ có thời gian chờ vô hạn, cho đến khi nó được cấp khóa. Có thể thay đổi điều này bằng cách thực thi lệnh SET LOCK_TIMEOUT. Lệnh sau đây đặt thời gian hết hạn cho một lệnh là 1000 mili giây (1 giây):

```
SET LOCK_TIMEOUT 1000
```

Như trong Chương trình 6.7, nếu đặt lệnh trên cho Giao dịch 2 thì khi phải chờ hơn 1 giây, SQL Server sẽ trả về lỗi và tự động hủy bỏ lệnh SQL này. ADO .Net cũng cho phép thiết lập thời gian hủy bỏ bằng cách thực thi lệnh SET LOCK_TIMEOUT như sau:

```
mySqlCommand.CommandText = "SET LOCK_TIMEOUT 1000";  
mySqlCommand.ExecuteNonQuery();
```

6.5.3. Chặn và đọc các giao dịch kiểu Serializable/Repeatable trong ADO .Net

Các giao dịch kiểu Serializable và Repeatable có tác dụng khóa các dòng mà chúng đang nhận để các giao dịch khác không thể cập nhật được. Các giao dịch Serializable và Repeatable được thực hiện để bảo đảm rằng các dòng sẽ không bị thay đổi sau khi chúng được đọc.

Chương trình 6.8 biểu diễn ví dụ về trường hợp khóa giao dịch. Nếu lấy các dòng từ bảng Account bằng cách dùng giao dịch có kiểu là Serializable và sau đó cố gắng cập nhật các dòng này bởi một giao dịch khác thì Giao dịch thứ 2 này sẽ bị chặn. Trong chương trình 6.8 Giao dịch 2 có thời gian chờ hủy bỏ là 1 giây. Khi thời gian chờ của giao dịch 2 vượt quá 1 giây, chương trình sẽ phát ra lỗi SqlException. Thực tế, khi chạy chương trình sẽ phát sinh lỗi, điều này nằm trong dự kiến. Nếu che đi lời gọi hàm DisplayRows đầu tiên, chương trình sẽ không phát ra lỗi bởi vì việc che đi lời gọi hàm DisplayRows() sẽ chặn giao dịch Serializable khỏi việc lấy dữ liệu và khóa các dòng.

Chương trình 6.8. Chặn và đọc giao dịch kiểu Serializable và Repeatable

```
public class LockTransaction  
{  
    /// <summary>  
    /// Phương thức đọc và hiển thị dữ liệu  
    /// </summary>  
    /// <param name="mySqlCommand">Đối tượng SqlCommand</param>  
    public static void DisplayRows(SqlCommand mySqlCommand)  
    {  
        mySqlCommand.CommandText = "Select * from Account where  
AccountName in ('tk1','quytd')";  
        SqlDataReader mySqlDataReader = mySqlCommand.ExecuteReader();  
        while (mySqlDataReader.Read())  
        {  
            Console.WriteLine("mySqlDataReader[\"AccountName\"] = " +  
mySqlDataReader["AccountName"]);  
            Console.WriteLine("mySqlDataReader[\"Full name\"] = " +  
mySqlDataReader["FullName"]);  
        }  
    }  
}
```

```
        }
        mySqlDataReader.Close();
    }
    public static void Main()
    {
        // Chuỗi kết nối
        string strConn = "server=localhost;database=RestaurantManagement;
integrated security=true;";
        // Tạo ra 2 đối tượng dùng để thực hiện giao dịch
        SqlConnection serConnection = new SqlConnection(strConn);
        serConnection.Open();
        SqlConnection rcConnection = new SqlConnection(strConn);
        rcConnection.Open();
        // Giao dịch 1 được tạo bằng cách tạo đối tượng SqlTransaction với
        // kiểu Serializable
        SqlTransaction serializableTrans =
            serConnection.BeginTransaction(IsolationLevel.Serializable);
        // Tạo một đối tượng SqlCommand và gán thuộc tính Transaction cho
        // Giao dịch 1
        SqlCommand serializableCommand = serConnection.CreateCommand();
        serializableCommand.Transaction = serializableTrans;
        // Gọi phương thức DisplayRows() để hiển thị số dòng từ bảng
        Account
        // việc này sẽ làm cho các dòng bị khóa khi thực hiện Giao dịch 2
        DisplayRows(serializableCommand);
        // Giao dịch 2 được tạo bằng cách tạo đối tượng SqlTransaction với
        // kiểu ReadCommitted
        SqlTransaction readCommittedTrans =
            rcConnection.BeginTransaction(IsolationLevel.ReadCommitted);
        // Tạo một đối tượng SqlCommand và gán thuộc tính Transaction cho
        // Giao dịch 2
        SqlCommand readCommittedCommand = rcConnection.CreateCommand();
        readCommittedCommand.Transaction = readCommittedTrans;
        // Thiết lập thời gian chờ là 1 giây
        readCommittedCommand.CommandText = "SET LOCK_TIMEOUT 1000";
        readCommittedCommand.ExecuteNonQuery();
        try
        {
            // Chèn 1 dòng mới vào bảng Account
            readCommittedCommand.CommandText =
                "Insert into Account (AccountName, Password, FullName) " +
                "Values ('tk1', '12345@', N'Nguyễn Văn A')";
            int numberOfRows = readCommittedCommand.ExecuteNonQuery();
            Console.WriteLine("Số dòng đã chèn = " + numberOfRows);
            // update the ALFKI row in the Customers table
            Console.WriteLine("Cập nhật thông tin tài khoản sẵn có");
            readCommittedCommand.CommandText =
                "UPDATE Account Set Email = 'thaiquyquy@gmail.com' " +
                "WHERE AccountName = 'quytd'";
            numberOfRows = readCommittedCommand.ExecuteNonQuery();
            Console.WriteLine("Số dòng cập nhật = " + numberOfRows);
            // display the new rows and rollback the changes
            DisplayRows(readCommittedCommand);
        }
```

```
        Console.WriteLine("Hủy bỏ Giao dịch 2");
        readCommittedTrans.Rollback();
    }
    catch (SqlException e) { Console.WriteLine(e); }
    finally { serConnection.Close(); rcConnection.Close(); }
    Console.ReadLine();
}
}
```

6.5.4. Deadlocks

Một deadlock xảy ra khi có hai giao dịch cùng chờ khóa mà một giao dịch khác đang chiếm giữ. Xét hai giao dịch (T1, T2) như trong Chương trình 6.9: Cả hai giao dịch đều tác động lên bảng Food và Table. Nếu cả hai được thực thi tại hai thời điểm khác nhau thì không xảy ra vấn đề gì. Tuy nhiên, nếu thực hiện đồng thời với các lệnh Update này, khi đó tình trạng deadlock sẽ xảy ra.

Chương trình 6.9. Hai giao dịch dùng lệnh Update sẽ xảy ra deadlocks

<pre>use RestaurantManagement --Giao dịch 1 (T1) BEGIN TRANSACTION -- Cập nhật dữ liệu bảng Food update Food set [Name] = 'ABCD' where ID = 1 --và bảng Table update [Table] set [Name] = 'VIP' where ID = 1 -- Thực thi lệnh cập nhật COMMIT TRANSACTION</pre>	<pre>use RestaurantManagement --Giao dịch 2 (T2) BEGIN TRANSACTION -- Cập nhật dữ liệu bảng Table update [Table] set [Name] = 'VIP' where ID = 1 --và bảng Table update Food set [Name] = 'ABCD' where ID = 1 -- Thực thi lệnh cập nhật COMMIT TRANSACTION</pre>
---	--

Xét Chương trình 6.9, ban đầu T1 và T2 chạy sẽ khóa dòng lần lượt trong bảng Food và bảng Table để tiến hành thực thi lệnh cập nhật trên dòng đó. Tiếp theo, T2 sẽ chờ để lấy khóa đang giữ bởi T1 trên dòng của bảng Food, ngược lại T1 lại chờ để lấy khóa đang giữ bởi T2 trên dòng của bảng Table. Lúc này, xảy ra tình trạng cả hai giao dịch đang chờ đợi lẫn nhau. SQL Server phát hiện ra tình trạng này và sẽ hủy bỏ một giao dịch (tốn ít chi phí nhất và trả về một lỗi cho biết tình trạng deadlocks).

Ta cũng có thể một chọn giao dịch để loại bỏ khi xảy ra tình trạng deadlock bằng cách dùng lệnh như sau:

```
SET DEADLOCK_PRIORITY LOW
```

Với ADO .Net, có thể gán độ ưu tiên của deadlocks bằng cách thiết lập lệnh cho đối tượng SqlCommand:

```
sqlCommand.CommandText = "SET DEADLOCK_PRIORITY LOW";
t2Command.ExecuteNonQuery();
```

Có thể giảm nguy cơ deadlock trong ADO .Net bằng cách tạo ra các giao dịch càng ngắn càng tốt. Với phương pháp này, thời gian mà các khóa bị chiếm giữ trên các đối tượng của cơ sở dữ liệu là ngắn nhất có thể. Chương trình 6.10 sau minh họa tình huống xảy ra deadlock khi có hai giao dịch T1 và T2 cùng nắm giữ khóa như được trình bày trong Chương trình 6.9.

Chương trình 6.10. Tình huống xảy ra deadlock với 2 giao dịch trong ADO .Net

```
public class Deadlock
{
    // Chuỗi kết nối
    public static string strConn =
"server=localhost;database=RestaurantManagement; integrated
security=true;";
    // Các đối tượng dùng cho Giao dịch 1
    public static SqlConnection t1Conn = new SqlConnection(strConn);
    public static SqlTransaction t1Trans;
    public static SqlCommand t1Command;
    // Các đối tượng dùng cho Giao dịch 2
    public static SqlConnection t2Conn = new SqlConnection(strConn);
    public static SqlTransaction t2Trans;
    public static SqlCommand t2Command;

    /// <summary>
    /// Phương thức thực thi lệnh cập nhật Food của Giao dịch 1
    /// </summary>
    public static void UpdateFoodT1()
    {
        String sql = "update Food set [Name] = 'ABCD' where ID = 1";
        t1Command.CommandText = sql;
        int numberOfRows = t1Command.ExecuteNonQuery();
        Console.WriteLine("So dong da cap nhat = " + numberOfRows);
    }
    /// <summary>
    /// Phương thức thực thi lệnh cập nhật Table của Giao dịch 1
    /// </summary>
    public static void UpdateTableT1()
    {
        String sql = "update [Table] set [Name] = 'VIP' where ID = 1";
        t1Command.CommandText = sql;
        int numberOfRows = t1Command.ExecuteNonQuery();
        Console.WriteLine("So dong da cap nhat = " + numberOfRows);
    }

    /// <summary>
    /// Phương thức thực thi lệnh cập nhật Table của Giao dịch 2
    /// </summary>
    public static void UpdateTableT2()
    {
        String sql = "update [Table] set [Name] = 'VIP' where ID = 1";
        t2Command.CommandText = sql;
        int numberOfRows = t2Command.ExecuteNonQuery();
        Console.WriteLine("So dong da cap nhat = " + numberOfRows);
    }
}
```

```
}

/// <summary>
/// Phương thức thực thi lệnh cập nhật Food của Giao dịch 2
/// </summary>
public static void UpdateFoodT2()
{
    String sql = "update Food set [Name] = 'ABCD' where ID = 1";
    t2Command.CommandText = sql;
    int numberOfRows = t2Command.ExecuteNonQuery();
    Console.WriteLine("Số dòng đã cập nhật = " + numberOfRows);
}

public static void Main()
{
    // Mở kết nối, bắt đầu giao dịch 1, thời gian khóa là 5s
    t1Connection.Open();
    t1Trans = t1Conn.BeginTransaction();
    t1Command = t1Conn.CreateCommand();
    t1Command.Transaction = t1Trans;
    t1Command.CommandText = "SET LOCK_TIMEOUT 5000";
    t1Command.ExecuteNonQuery();

    // Mở kết nối, bắt đầu giao dịch 2, thời gian khóa là 5s
    t2Connection.Open();
    t2Trans = t2Conn.BeginTransaction();
    t2Command = t2Conn.CreateCommand();
    t2Command.Transaction = t2Trans;
    t2Command.CommandText = "SET LOCK_TIMEOUT 5000";
    t2Command.ExecuteNonQuery();

    // Thiết lập DEADLOCK_PRIORITY LOW cho giao dịch thứ hai
    // điều này làm cho giao dịch 2 bị hủy bỏ
    t2Command.CommandText = "SET DEADLOCK_PRIORITY LOW";
    t2Command.ExecuteNonQuery();

    // Tạo ra 4 luồng để thực thi đồng thời
    Thread FoodT1 = new Thread(new ThreadStart(UpdateFoodT1));
    FoodT1.Start();
    Thread TableT2 = new Thread(new ThreadStart(UpdateTableT2));
    TableT2.Start();
    Thread TableT1 = new Thread(new ThreadStart(UpdateTableT1));
    TableT1.Start();
    Thread FoodT2 = new Thread(new ThreadStart(UpdateFoodT2));
    FoodT2.Start();
}

}
```

Chương trình 6.10 sử dụng Thread như là các tiến trình chạy độc lập với nhau, mỗi Thread thực thi các lệnh một cách song song với các Thread khác. Các phương thức được gọi bởi các Thread để thực hiện việc cập nhật xen kẽ nhau. Chương trình 6.10 cũng cho biết giao dịch T2 sẽ bị hủy bỏ khi có xảy ra deadlock vì có dùng đến lệnh SET DEADLOCK_PRIORITY LOW.

6.6. Kết chương

Ngày nay, các cơ sở dữ liệu có thể xử lý việc có nhiều người dùng và nhiều chương trình cùng truy xuất tới cơ sở dữ liệu tại một thời điểm. Mỗi người dùng hay chương trình đều có khả năng thực hiện các giao dịch của mình với cơ sở dữ liệu. Phần mềm quản trị cơ sở dữ liệu phải có khả năng đáp ứng được các yêu cầu cho tất cả các giao dịch, đồng thời cũng như duy trì được tính toàn vẹn của dữ liệu được lưu trong các bảng. Chúng ta có thể điều khiển mức độ độc lập của giao dịch đang tồn tại với các giao dịch khác đang chạy trong cơ sở dữ liệu.

Chương này cũng đi sâu vào việc kiểm soát các giao dịch bằng cách sử dụng lệnh trong SQL Server và ADO .Net, cách thiết lập điểm lưu (*savepoint*), quay ngược giao dịch tới điểm lưu và thiết lập mức độ độc lập của giao dịch. Chúng ta cũng tìm hiểu về các chế độ khóa của SQL Server và cách ngăn chặn các giao dịch cũng như làm giảm nguy cơ xảy ra deadlock bởi các giao dịch khác.

Bài tập

1. Xây dựng Transaction với SQL Server và ADO .Net trên Form
2. Sử dụng Thread trên Windows Form, xây dựng minh họa với các giao dịch Phantom, Nonrepeatable reads, và Dirty reads.
3. Xây dựng minh họa trên Form các phương pháp chặn giao dịch.

CHƯƠNG 7. MÔ HÌNH ĐA TẦNG

Mô hình Client/Server, mô hình OSI, mô hình MVC, hay mô hình *n-tier* (đa tầng) là những thuật ngữ mà chúng ta từng nghe tới khi tìm hiểu về một ứng dụng trong lĩnh vực công nghệ thông tin. Mỗi mô hình có những đặc điểm riêng dùng để áp dụng cho một lĩnh vực nào đó. Ứng dụng sẽ hoạt động hiệu quả, dễ nâng cấp, bảo trì nếu được lựa chọn mô hình phù hợp, ngược lại sẽ khó quản lý, bảo hành nếu áp dụng một mô hình không phù hợp. Do vậy, việc lựa chọn mô hình phù hợp cho ứng dụng rất quan trọng.

Không như phương pháp lập trình trước đây là viết chương trình và chạy theo các dòng lệnh. Ngày nay, khi phát triển các mô hình lập trình người ta thường đề cập đến khái niệm mô hình đa tầng. Đây là một khái niệm tuy không mới, nhưng vẫn còn được ứng dụng nhiều trong phương pháp lập trình phần mềm. Việc áp dụng theo mô hình đa tầng nhằm giúp cho người lập trình dễ dàng quản lý được mã nguồn, bảo trì, bảo hành theo từng phần, đồng thời giúp cho tính kế thừa được thực thi một cách thuận lợi.

Chương này nhằm mục đích giới thiệu về mô hình đa tầng (*n-tier*) trong các phương pháp lập trình ứng dụng hiện nay. Các ví dụ minh họa trong chương này sử dụng ngôn ngữ C# với bộ Visual Studio và SQL Server. Các nội dung chính của chương bao gồm:

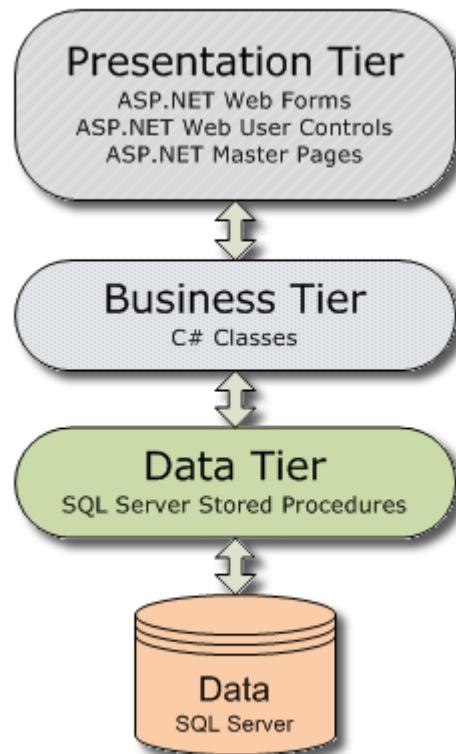
- Giới thiệu mô hình đa tầng;
- Kiến trúc mô hình đa tầng;
- Tầng truy cập dữ liệu (*Data Access - DA*);
- Tầng xử lý logic (*Business Logic - BL*);
- Tầng giao diện người dùng (*User Interface - UI*).

7.1. Giới thiệu mô hình đa tầng

Khi phát triển một chương trình, điều thường gây khó khăn nhất cho người lập trình là quá trình kế thừa, bảo trì và bảo hành khi có lỗi hoặc phát sinh các nghiệp vụ mới. Các phương pháp lập trình theo dòng lệnh hoặc theo hướng đối tượng nếu dự án đơn giản thì việc bảo trì hay bảo hành là công việc không khó khăn. Tuy nhiên, khi dự án đã trở nên lớn với rất nhiều dòng lệnh và nhiều lớp thì công việc này trở nên phức tạp. Do đó, trong quá trình phát triển phần mềm người ta đã sinh ra khái niệm mô hình đa tầng, đa lớp (tiếng Anh là *n-tier*) nhằm tối ưu việc tách biệt các tầng, mỗi tầng một nhiệm vụ để dễ dàng quản lý, bảo trì, bảo hành sau này.

Vậy mô hình đa tầng (*n-tier*) là một kiến trúc phần mềm được phân chia thành nhiều các thành phần, trong đó phần giao diện người dùng (*User Interface - UI*), các quy tắc xử lý (*Business Logic - BL*), và mô hình dùng để quản lý, lưu trữ dữ liệu (*Data Access - DA*) được phát triển như là những mô-đun độc lập. Các mô-đun này có thể được xây dựng và phát triển trên các nền tảng độc lập, chúng được nối với nhau thông qua việc giao tiếp qua các tập tin dạng thư viện (*.dll). Mô hình đa tầng được coi là một kiến trúc phần mềm và là một dạng của mẫu thiết kế (*design pattern*).

Khi triển khai ứng dụng ở mức vật lý, kiến trúc đa tầng thường đưa về dạng kiến trúc với ba tầng riêng biệt bao gồm: Tầng giao diện (*Presentation*), tầng xử lý (*Business Logic*) và tầng truy cập dữ liệu (*Data Access*) (Hình 7.1). Tầng giao diện dùng để hiển thị các thành phần giao diện và tương tác với người dùng như tiếp nhận thông tin nhập, thông báo kết quả, thông báo lỗi... Tầng xử lý thực hiện các hành động nghiệp vụ của phần mềm như tính toán, thêm, xóa, sửa dữ liệu... Tầng này còn có vai trò là trung gian giữa hai tầng trên và dưới. Tầng truy xuất dữ liệu thường bao gồm hai thành phần: Thành phần trực tiếp truy xuất dữ liệu thường là các lệnh, các hàm trong hệ quản trị cơ sở dữ liệu; Thành phần truy xuất là các lớp tổng quát dùng để gọi các hàm và lệnh từ cơ sở dữ liệu.



Hình 7.1. Mô hình ba tầng đơn giản

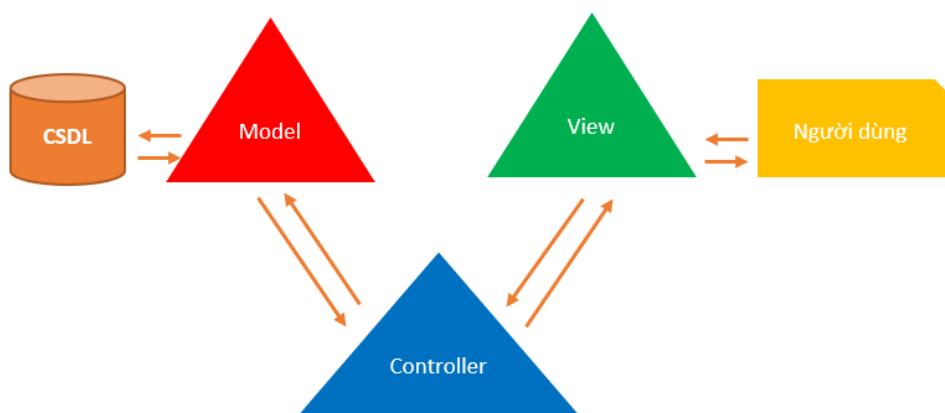
Nguồn: [https://medium.com/@hidayatarg/n-tier-layer-architecture-in-c-15b8fe97283c#:~:text=N%20represent%20a%20number%20and,\(Layers\)%20of%20an%20application.&text=It%20comprise%20of%203%2Dtiers,ands%20Layers%20are%20used%20interchangeably](https://medium.com/@hidayatarg/n-tier-layer-architecture-in-c-15b8fe97283c#:~:text=N%20represent%20a%20number%20and,(Layers)%20of%20an%20application.&text=It%20comprise%20of%203%2Dtiers,ands%20Layers%20are%20used%20interchangeably).

Mọi người vẫn hay nhầm lẫn giữa *tier* và *layer* vì cấu trúc giống nhau. Tuy nhiên, thực tế thì *tier* chia theo tính chất vật lý còn *layer* lại có tính logic. Nghĩa là ta chia ứng dụng thành các tầng để xử lý, trong các tầng đó có các lớp khác nhau phụ trách các mảng khác nhau. Sau đây là một số ưu điểm của mô hình đa tầng:

- Việc phân chia thành từng tầng giúp cho việc viết mã nguồn được tách minh hơn. Mỗi tầng sẽ đảm nhận một chức năng khác nhau như giao diện, xử lý thay vì để tất cả trong một chỗ;

- Khi cần phải bảo trì, việc thay đổi một vài thành phần sẽ dễ dàng hơn, có thể xác định hoặc cô lập ngay trong một tầng mà không ảnh hưởng tới toàn bộ chương trình;
- Khi muốn phát triển thêm một chức năng thì việc lập trình theo mô hình sẽ dễ dàng hơn vì đã cho một quy chuẩn cho trước. Một ví dụ là nếu thay đổi giao diện giữa Web và Form thì chỉ cần thay đổi tầng giao diện, các tầng còn lại có thể giữ nguyên;
- Công việc tương tác, làm việc nhóm và bàn giao tác vụ sẽ dễ dàng hơn với mô hình đa tầng. Mỗi nhóm, mỗi bộ phận sẽ đảm nhiệm một công việc nhất định, các tầng có thể được giao độc lập cho các nhóm, khi đó sự tương tác giữa các tầng sẽ thông qua các tập tin thư viện (*.dll).

Một trong những mô hình đa tầng nổi tiếng mà các nhà lập trình hay sử dụng trong lập trình Web đó là mô hình MVC với kiến trúc ba tầng là Model – View – Controller (Hình 7.2). Mô hình này là sự kế thừa của mô hình đa tầng, tầng Model biểu diễn các mô hình minh họa lại các bảng trong cơ sở dữ liệu, tầng View biểu diễn và xử lý giao diện, còn tầng Controller là các lớp trung gian điều khiển và xử lý từ cơ sở dữ liệu lên giao diện.



Hình 7.2. Mô hình MVC

7.2. Tạo ứng dụng mô hình đa tầng trên Visual Studio

Sử dụng Visual Studio, chúng ta có thể tạo chương trình với mô hình ba tầng như sau:

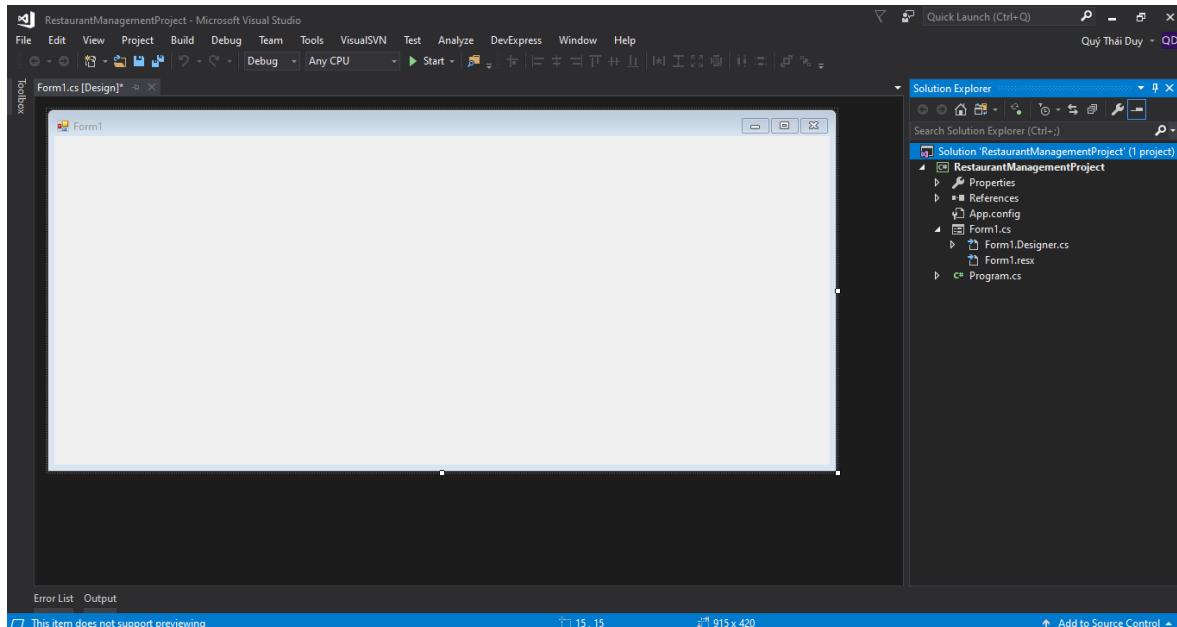
- Tầng giao diện (*GUI*): Tầng này có thể tạo bằng Web hoặc Form, dùng để xử lý giao diện người dùng. Các thao tác như nhấp chuột hay nhấn nút được thực hiện trên tầng này. Thường thì tầng này được tạo trước;
- Tầng xử lý (*Business Logic*): Tầng này dùng để tạo các phương thức xử lý trung gian. Các phương thức trong tầng này là trung gian giữa hai tầng còn lại, nhận dữ liệu từ tầng dưới, xử lý và đưa kết quả lên tầng trên.

- Tầng truy cập dữ liệu (*Data Access*): Lớp này bao gồm hai thành phần: Thành phần xử lý trong cơ sở dữ liệu và thành phần xử lý trên mã nguồn. Thành phần trong cơ sở dữ liệu chính là các bảng, các Store Procedure, Function...; Thành phần trong mã nguồn bao gồm mô hình ánh xạ bảng, các phương thức tổng quát để truy xuất dữ liệu (Mục 7.3).

Khi xây dựng mô hình đa tầng, ta cần có một Solution cho toàn bộ dự án, thường thì Solution được tạo ra khi tạo tầng đầu tiên (*Web, Form*). Các tầng còn lại có thể thêm mới hoặc thêm từ dự án có sẵn (tập tin *.dll). Các tầng được liên kết với nhau thông qua việc kết nối bởi các tập tin thư viện.

Ví dụ: Để tạo mô hình đa tầng với cơ sở dữ liệu Quản lý nhà hàng, ta thực hiện theo các bước sau:

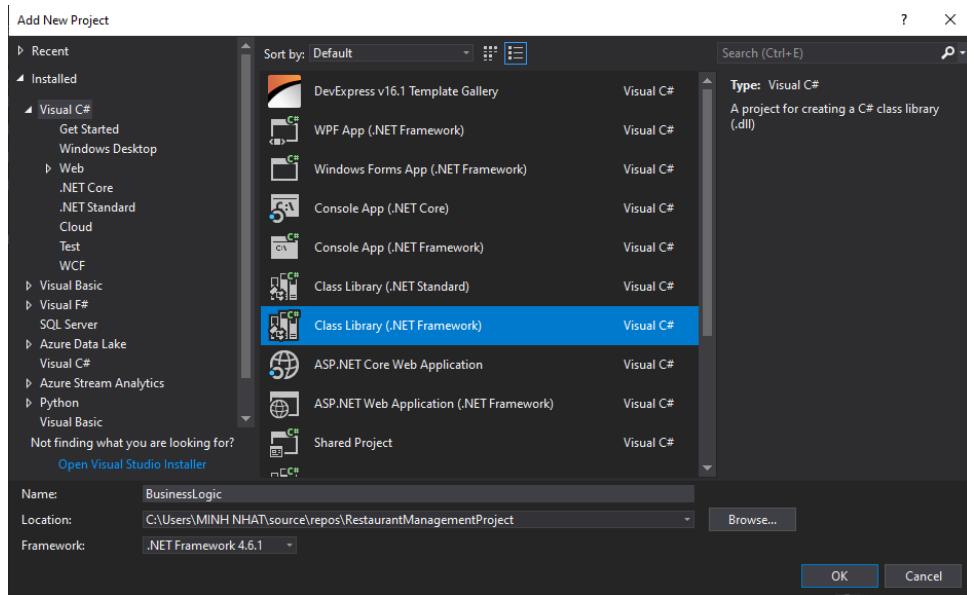
- Bước 1: Mở Visual Studio, tạo dự án Windows Form bằng ngôn ngữ C#, đặt tên dự án này là *RestaurantManagementProject* (Hình 7.3). Đây chính là tầng Giao diện, dùng để xử lý các vấn đề liên quan đến giao diện chương trình.



Hình 7.3. Tạo một dự án từ Visual Studio

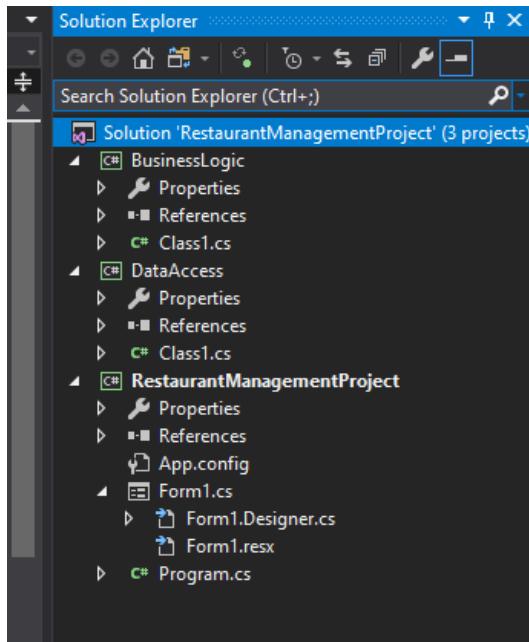
Lưu ý: Sau khi tạo xong dự án, ta thấy có một Solution và một Project được tạo ra. Từ đây, Solution là giải pháp chung cho cả dự án, mỗi một Project đại diện cho một tầng.

- Bước 2: Tạo tầng xử lý bằng cách click phải lên Solution, chọn Add, chọn New Project. Trong hộp thoại hiện lên chọn kiểu dự án là Class Library (.Net Frame work), đặt tên dự án là *BusinessLogic* (Hình 7.4). Đây là một dự án dạng thư viện, thư viện này được tạo ra dưới dạng các tập tin *.dll khi chạy ứng dụng.



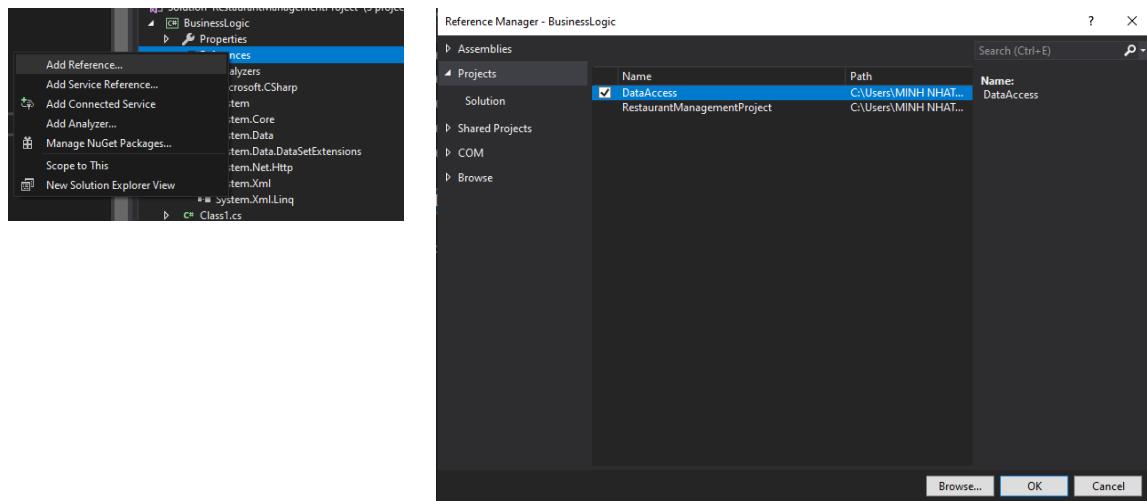
Hình 7.4. Tạo tầng xử lý trong Visual Studio

- *Bước 3:* Thực hiện thao tác tương tự Bước 2 để tạo tầng truy cập dữ liệu có tên là DataAccess. Kết quả trên Solution Explorer được thể hiện như trong Hình 7.5.



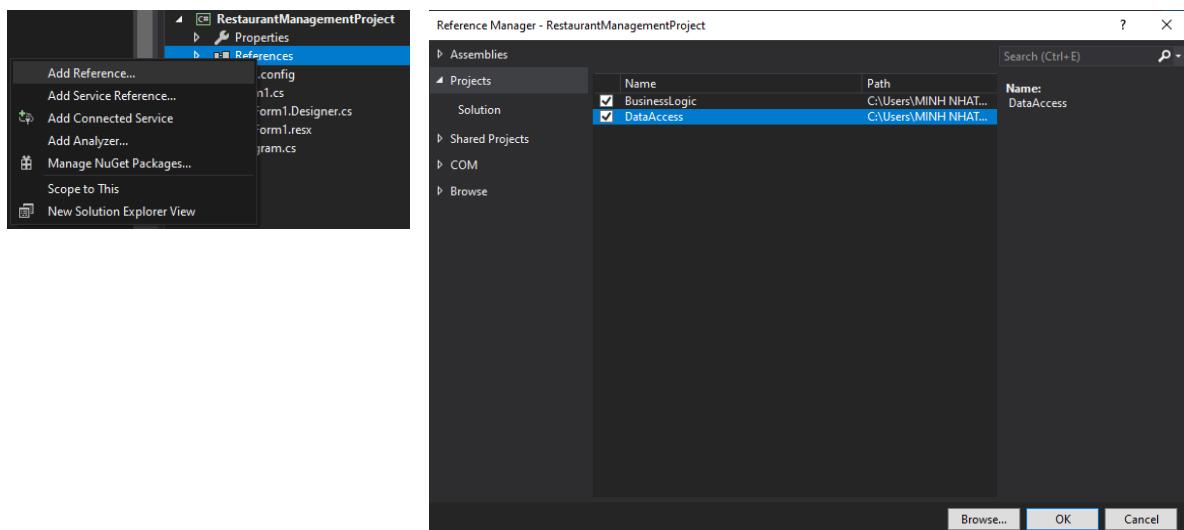
Hình 7.5. Mô hình ba tầng khi tạo dự án trên Visual Studio

- *Bước 4:* Để có thể gọi các phương thức của tầng DataAccess từ tầng BusinessLogic, ta click phải lên thành phần References của Project BusinessLogic, chọn Add Reference..., đánh dấu tích và DataAccess (Hình 7.6).



Hình 7.6. Kết nối tầng BusinessLayer và tầng DataAccess

Thực hiện tương tự để kết nối hai tầng BusinessLayer và DataAccess với tầng giao diện (Hình 7.7).



Hình 7.7. Kết nối hai tầng BusinessLayer và DataAccess với tầng Giao diện

Sau khi đã kết nối các tầng, chúng ta có thể gọi các đối tượng, phương thức, và thuộc tính từ các tầng với nhau bằng cách sử dụng lệnh using như sau:

```
using BusinessLogic; hoặc using DataAccess;
```

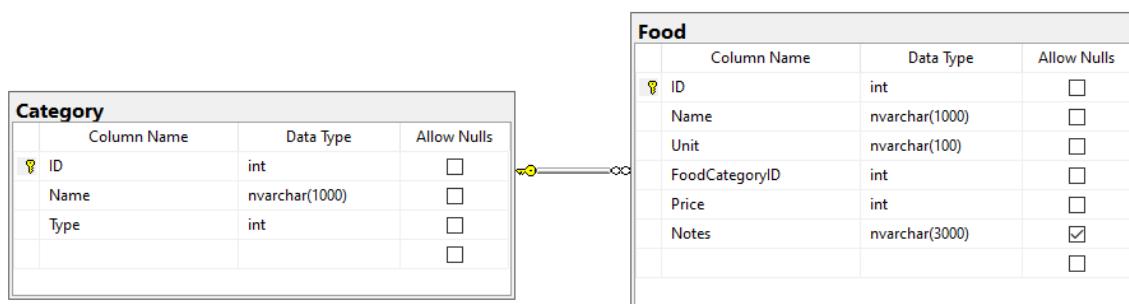
Chi tiết về cách sử dụng các tầng này có thể xem tại các mục tiếp theo của chương này. Thứ tự xây dựng các tầng tuân theo quy tắc tầng nào có tính tổng quát thì xây dựng trước, tầng nào chi tiết thì xây dựng sau. Theo quy tắc trên thì tầng DataAccess được xây dựng trước, tới tầng BusinessLogic, và cuối cùng là tầng Presentation.

7.3. Tầng truy cập dữ liệu (DataAccess)

Tầng DataAccess bao gồm hai thành phần là Cơ sở dữ liệu (*Data*) và phần Truy xuất (*Access*). Phần cơ sở dữ liệu được quy ước để sau này các lớp trên truy xuất một cách dễ dàng, phần truy xuất sẽ xây dựng một số lớp và phương thức ở mức tổng quát để truy xuất dữ liệu. Ở đây sử dụng dự án đã tạo ở Mục 7.2 để thực hiện xây dựng phần DataAccess.

7.3.1. Xây dựng cơ sở dữ liệu

Cơ sở dữ liệu bao gồm tên cơ sở dữ liệu, các bảng, các thủ tục và các hàm. Khi xây dựng mô hình đa tầng cần nhất quán một số vấn đề về tên gọi để sau này dễ phân biệt. Ở đây, cơ sở dữ liệu được sử dụng là RestaurantManagement, hai bảng được dùng để minh họa là Category và Food, đây là hai bảng có kết nối khóa ngoại, có giá trị ID tự tăng làm khóa chính (Hình 7.8).



Hình 7.8. Mối quan hệ của bảng Category và Food

Để dễ dàng minh họa việc thêm, xoá, sửa và lấy dữ liệu, mỗi bảng tầng này sẽ được xây dựng hai thủ tục (SprocureProcedure) là Lấy tất cả (GetAll) và Thêm_Xoá_Sửa (InsertUpdateDelete).

Bảng 7.1. Các thủ tục (SprocureProcedure) chính của tầng DataAccess trong SQL Server

Tên thủ tục	Kiểu trả về	Giải thích
Category_GetAll	Toàn bộ mẫu tin bảng Category	Thủ tục này trả về tất cả mẫu tin trong bảng Category.
Category_InsertUpdateDelete	Kiểu số nguyên	Thủ tục này nhận vào các tham số bảng Category, và biến action, nếu action = 0 thì thêm, nếu bằng 1 thì sửa và nếu bằng 2 thì xoá.
Food_GetAll	Toàn bộ mẫu tin bảng Food	Thủ tục này trả về tất cả mẫu tin trong bảng Food.
Food_InsertUpdateDelete	Kiểu số nguyên	Thủ tục này nhận vào các tham số bảng Food, và biến action, nếu action = 0 thì thêm, nếu bằng 1 thì sửa và nếu bằng 2 thì xoá.

Chi tiết các thủ tục được mô tả như trong Bảng 7.1, chi tiết về mã lệnh của các thủ tục có ở Chương trình 7.1, Chương trình 7.2, và Chương trình 7.3. Thủ tục InsertUpdateDelete có thể được tách thành ba thủ tục nhỏ hơn để dễ xử lý. Trong tầng này có thể thêm một số thủ tục khác như Lấy mẫu tin theo khoá (GetByID), Tìm kiếm (FindBy)... tuy nhiên các phương thức này hoàn toàn có thể được xử lý dựa trên tầng Business nên ở đây không đưa thêm vào.

Chương trình 7.1. Thủ tục GetAll của bảng Category và bảng Food

-- Thủ tục lấy tất cả dữ liệu bảng Category CREATE PROCEDURE [dbo].[Category_GetAll] AS SELECT * FROM Category	-- Thủ tục lấy tất cả dữ liệu bảng Food ALTER PROCEDURE [dbo].[Food_GetAll] AS SELECT * FROM Food
---	--

Chương trình 7.2. Thủ tục InsertUpdateDelete của bảng Category

```
USE [RestaurantManagement]
GO
-- Thủ tục thêm, xóa, sửa bảng Category
ALTER PROCEDURE [dbo].[Category_InsertUpdateDelete]
    @ID int output, -- Biến ID tự tăng, khi thêm xong phải lấy ra
    @Name nvarchar(200),
    @Type int,
    @Action int -- Biến cho biết thêm, xóa, hay sửa
AS
-- Nếu Action = 0, thực hiện thêm dữ liệu
IF @Action = 0
    BEGIN
        INSERT INTO [Category] ([Name], [Type])
        VALUES (@Name, @Type)
        SET @ID = @@identity -- Thiết lập ID tự tăng
    END
-- Nếu Action = 1, thực hiện cập nhật dữ liệu
ELSE IF @Action = 1
    BEGIN
        UPDATE [Category] SET [Name] = @Name, [Type]=@Type
        WHERE [ID] = @ID
    END
-- Nếu Action = 2, thực hiện xóa dữ liệu
ELSE IF @Action = 2
    BEGIN
        DELETE FROM [Category] WHERE [ID] = @ID
    END
```

Chương trình 7.3. Thủ tục InsertUpdateDelete của bảng Food

```
USE [RestaurantManagement]
GO
-- Thủ tục thêm, xóa, sửa bảng Food
ALTER PROCEDURE [dbo].[Food_InsertUpdateDelete]
    @ID int output, -- Biến ID tự tăng, khi thêm xong phải lấy ra
```

```
@Name nvarchar(1000),
@Unit nvarchar(100),
@FoodCategoryID int,
@Price int,
@Notes nvarchar(3000),
@Action int -- Biến cho biết thêm, xóa, hay sửa
AS
-- Nếu Action = 0, thực hiện thêm dữ liệu
IF @Action = 0
    BEGIN
        INSERT INTO [Food]
        ([Name],[Unit],[FoodCategoryID],[Price],[Notes])
        VALUES (@Name, @Unit,@FoodCategoryID,@Price,@Notes)
        SET @ID = @@identity -- Thiết lập ID tự tăng
    END
-- Nếu Action = 1, thực hiện cập nhật dữ liệu
ELSE IF @Action = 1
    BEGIN
        UPDATE [Food]
        SET [Name] = @Name,[Unit]=@Unit,[FoodCategoryID]=@FoodCategoryID,
            [Price]=@Price,[Notes]=@Notes
        WHERE [ID] = @ID
    END
-- Nếu Action = 2, thực hiện xóa dữ liệu
ELSE IF @Action = 2
    BEGIN
        DELETE FROM [Food] WHERE [ID] = @ID
    END
```

7.3.2. Xây dựng lớp các tham số chung

Lớp các tham số chung dùng để khai báo các tham số dùng chung cho phần DataAccess. Các tham số dùng chung bao gồm một chuỗi kết nối dùng để kết nối cơ sở dữ liệu lấy từ tập tin App.config (xem Mục 7.5.1) và các tham số là tên của các thủ tục trong cơ sở dữ liệu. Ngoài ra, khi phát triển chương trình có thể thêm các tham số khác. Click phải lên Project DataAccess, chọn Add, chọn Class, đặt tên lớp là Utilities, viết mã nguồn như trong Chương trình 7.4. Để gọi được lớp ConfigurationManager cần phải thêm thư viện System.Configuration vào dự án (click phải lên References, chọn Add Reference..., tìm đến thư viện System.Configuration và thêm vào dự án), sau đó gọi thư viện này bằng lệnh using:

```
using System.Configuration;
```

Chương trình 7.4. Mã nguồn lớp các tham số chung (Utilities)

```
public class Utilities
{
    // Chuỗi kết nối
    private static string StrName = "ConnectionStringName";
    public static string ConnectionString = ConfigurationManager
        .ConnectionStrings[StrName]
        .ConnectionString;
    // Các biến của bảng Food
```

```
public static string Food_GetAll = "Food_GetAll";
public static string Food_InsertUpdateDelete =
"Food_InsertUpdateDelete";
// Các biến của bảng Food
public static string Category_GetAll = "Category_GetAll";
public static string Category_InsertUpdateDelete=
"Category_InsertUpdateDelete";
}
```

7.3.3. Xây dựng các lớp ánh xạ bảng

Mỗi một bảng trong cơ sở dữ liệu sẽ có một lớp ánh xạ, lớp này bao gồm các thuộc tính và kiểu dữ liệu tương ứng như trong SQL Server. Bảng 7.2 là bảng ánh xạ kiểu dữ liệu SQL Server sang kiểu dữ liệu trong ngôn ngữ C#.

Bảng 7.2. Ánh xạ kiểu dữ liệu từ SQL Server sang C#

Kiểu trong SQL Server	Kiểu trong C#	Giải thích
char, nvarchar, text	String	Kiểu dữ liệu chuỗi
bit	Bool	Kiểu dữ liệu chỉ có 2 giá trị
Datetime, smalldatetime	DateTime	Kiểu dữ liệu dạng ngày tháng (hoặc giờ)
smallint	Int16	Kiểu Interger 16 bit
int	Int32	Kiểu Interger 32 bit
bigint	long	Kiểu Interger dài (64 bit)
float	float	Kiểu số thực ngắn
money, real	Double	Kiểu số thực dài
decimal	Decimal	Kiểu thập phân
Null	?	Ví dụ: int? là kiểu int cho phép null

Ví dụ ở đây sử dụng hai bảng là Food và Category nên lớp ánh xạ chính là hai bảng này. Click phải lên Project DataAccess, chọn Add, chọn Class, đặt tên lớp lần lượt là Food.cs và Category.cs. Khai báo các thuộc tính và kiểu dữ liệu tương ứng với các bảng trong SQL Server, chi tiết được mô tả như trong Chương trình 7.5 và Chương trình 7.6.

Chương trình 7.5. Mô hình ánh xạ bảng Category

```
public class Category
{
    // ID của bảng, tự tăng trong CSDL
    public int ID { get; set; }
    // Tên của loại thức ăn
    public string Name { get; set; }
    // Kiểu: 0 là đồ uống; 1 là thức ăn...
    public int Type { get; set; }
}
```

Chương trình 7.6. Mô hình ánh xạ bảng Food

```
public class FoodRecord
{
    // ID của bảng Food
    public int ID { get; set; }
    // Tên loại đồ ăn, thức uống
    public string Name { get; set; }
    // Đơn vị tính
    public string Unit { get; set; }
    // Loại thức ăn, ứng với bảng ở trên
    public int FoodCategoryID { get; set; }
    // Giá
    public int Price { get; set; }
    // Ghi chú
    public string Notes { get; set; }
}
```

7.3.4. Xây dựng các lớp truy xuất dữ liệu

Mỗi một thủ tục trong cơ sở dữ liệu sẽ có một phương thức tương ứng dùng để gọi. Các phương thức xử lý theo từng bảng sẽ được gom lại thành một lớp có cấu trúc Bảng_Tên thủ tục. Ví dụ: Bảng Category có hai thủ tục dùng để lấy tất cả (GetAll) và thêm, xoá, sửa mẫu tin (InsertUpdateDelete) thì sẽ có hai phương thức tương ứng là GetAll() và Insert_Update_Delete(...). Phương thức GetAll thì không truyền tham số và trả về là một danh sách các mô hình tương ứng, còn phương thức Insert_Update_Delete thì truyền vào một đối tượng là ánh xạ của bảng và một biến action. Chương trình 7.7 và Chương trình 7.8 minh họa các lớp của hai bảng là Category và Food với hai lớp tương ứng là CategoryDA và FoodDA.

Chương trình 7.7. Lớp truy xuất dữ liệu từ bảng Category

```
//Lớp quản lý Category: DA = DataAccess
public class CategoryDA
{
    // Phương thức lấy hết dữ liệu theo thủ tục Food_GetAll
    public List<Category> GetAll()
    {
        // Khai báo đối tượng SqlConnection và mở kết nối
        // Đối tượng SqlConnection truyền vào chuỗi kết nối trong
        App.config
        SqlConnection sqlConn=new
        SqlConnection(Ultilities.ConnectionString);
        sqlConn.Open();
        // Khai báo đối tượng SqlCommand có kiểu xử lý là StoredProcedure
        SqlCommand command = sqlConn.CreateCommand();
        command.CommandType = CommandType.StoredProcedure;
        command.CommandText = Ultilities.Category_GetAll;
        // Đọc dữ liệu, trả về danh sách các đối tượng Category
        SqlDataReader reader = command.ExecuteReader();
        List<Category> list = new List<Category>();
        while (reader.Read())
        {
```

```
        Category category = new Category();
        category.ID = Convert.ToInt32(reader["ID"]);
        category.Name = reader["Name"].ToString();
        category.Type = Convert.ToInt32(reader["Type"]);
        list.Add(category);
    }
    // Đóng kết nối và trả về danh sách
    sqlConn.Close();
    return list;
}
//Phương thức thêm, xoá, sửa theo thủ tục Category_InsertUpdateDelete
public int Insert_Update_Delete(Category category, int action)
{
    // Khai báo đối tượng SqlConnection và mở kết nối
    // Đối tượng SqlConnection truyền vào chuỗi kết nối trong
    App.config
    SqlConnection sqlConn=new
    SqlConnection(Ultilities.ConnectionString);
    sqlConn.Open();
    //Khai báo đối tượng SqlCommand có kiểu xử lý là StoredProcedure
    SqlCommand command = sqlConn.CreateCommand();
    command.CommandType = CommandType.StoredProcedure;
    command.CommandText = Ultilities.Category_InsertUpdateDelete;
    // Thêm các tham số cho thủ tục; Các tham số này chính là các tham
    // số trong thủ tục;
    //ID là tham số có giá trị lấy ra khi thêm và truyền vào khi xoá, sửa
    SqlParameter IDPara = new SqlParameter("@ID", SqlDbType.Int);
    IDPara.Direction = ParameterDirection.InputOutput; // Vào và ra
    command.Parameters.Add(IDPara).Value = category.ID;
    command.Parameters.Add("@Name", SqlDbType.NVarChar,200)
        .Value = category.Name;
    command.Parameters.Add("@Type", SqlDbType.Int)
        .Value = category.Type;
    command.Parameters.Add("@Action", SqlDbType.Int)
        .Value = action;
    // Thực thi lệnh
    int result = command.ExecuteNonQuery();
    if (result > 0) // Nếu thành công thì trả về ID đã thêm
        return (int)command.Parameters["@ID"].Value;
    return 0;
}
}
```

Chương trình 7.8. Lớp truy xuất dữ liệu từ bảng Food

```
//Lớp quản lý Food: DA = DataAccess
public class FoodDA
{
    // Phương thức lấy hết dữ liệu theo thủ tục Food_GetAll
    public List<Food> GetAll()
    {
        //Khai báo đối tượng SqlConnection và mở kết nối
        //Đối tượng SqlConnection truyền vào chuỗi kết nối trong
        App.config
```

```
SqlConnection sqlConn=new  
SqlConnection(Ultilities.ConnectionString);  
sqlConn.Open();  
//Khai báo đối tượng SqlCommand có kiểu xử lý là StoredProcedure  
SqlCommand command = sqlConn.CreateCommand();  
command.CommandType = CommandType.StoredProcedure;  
command.CommandText = Ultilities.Food_GetAll;  
// Đọc dữ liệu, trả về danh sách các đối tượng Food  
SqlDataReader reader = command.ExecuteReader();  
List<Food> list = new List<Food>();  
while (reader.Read())  
{  
    Food food = new Food();  
    food.ID = Convert.ToInt32(reader["ID"]);  
    food.Name = reader["Name"].ToString();  
    food.Unit = reader["Unit"].ToString();  
    food.FoodCategoryID =  
Convert.ToInt32(reader["FoodCategoryID"]);  
    food.Price = Convert.ToInt32(reader["Price"]);  
    food.Notes = reader["Notes"].ToString();  
    list.Add(food);  
}  
// Đóng kết nối và trả về danh sách  
sqlConn.Close();  
return list;  
}  
// Phương thức thêm, xoá, sửa theo thủ tục Food_InsertUpdateDelete  
public int Insert_Update_Delete(Food food, int action)  
{  
    // Khai báo đối tượng SqlConnection và mở kết nối  
    // Đối tượng SqlConnection truyền vào chuỗi kết nối trong  
App.config  
    SqlConnection sqlConn=new  
SqlConnection(Ultilities.ConnectionString);  
    sqlConn.Open();  
    //Khai báo đối tượng SqlCommand có kiểu xử lý là StoredProcedure  
    SqlCommand command = sqlConn.CreateCommand();  
    command.CommandType = CommandType.StoredProcedure;  
    command.CommandText = Ultilities.Food_InsertUpdateDelete;  
    // Thêm các tham số cho thủ tục; Các tham số này chính là các tham  
số trong thủ tục;  
    //ID là tham số có giá trị lấy ra khi thêm và truyền vào khi xoá,  
sửa  
    SqlParameter IDPara = new SqlParameter("@ID", SqlDbType.Int);  
    IDPara.Direction = ParameterDirection.InputOutput;  
    command.Parameters.Add(IDPara).Value = food.ID;  
    //Các biến còn lại chỉ truyền vào  
    command.Parameters.Add("@Name", SqlDbType.NVarChar, 1000)  
        .Value = food.Name;  
    command.Parameters.Add("@Unit", SqlDbType.NVarChar)  
        .Value = food.Unit;  
    command.Parameters.Add("@FoodCategoryID", SqlDbType.Int)  
        .Value = food.FoodCategoryID;
```

```
        command.Parameters.Add("@Price", SqlDbType.Int)
                    .Value = food.Price;
        command.Parameters.Add("@Notes", SqlDbType.NVarChar, 3000)
                    .Value = food.Notes;
        command.Parameters.Add("@Action", SqlDbType.Int)
                    .Value = action;
        int result = command.ExecuteNonQuery();
        // Thực thi lệnh
        if (result > 0) // Nếu thành công thì trả về ID đã thêm
            return (int)command.Parameters["@ID"].Value;
        return 0;
    }
}
```

Chương trình 7.7 và Chương trình 7.8 chỉ là hai chương trình đơn giản dùng để minh họa, thực tế thì khi viết dự án, có bao nhiêu thủ tục trong cơ sở dữ liệu thì phải viết bấy nhiêu phương thức tương ứng. Ngoài ra, các phương thức tổng quát hơn có thể được viết ở tầng này, để dễ hiểu, ở đây chỉ đưa vào một số phương thức đơn giản. Khi viết xong tầng này, có thể kiểm tra lỗi và tạo tập tin *.dll bằng cách click phải lên *DataAccess*, chọn *Build*, khi đó nếu có lỗi sai, hệ thống sẽ báo để sửa lỗi trước khi viết qua tầng mới.

7.4. Tầng xử lý logic (*BusinessLogic - BLL*)

Tầng *BusinessLogic* là tầng xử lý trung gian giữa tầng *DataAccess* và tầng *Presentation*. Tầng này bao gồm các lớp dạng *Bảng_BL*, dùng để xử lý chi tiết hơn tầng *DataAccess*. Tầng này có thể thêm một số phương thức khác như Tim kiếm, tìm theo khoá chính, tìm theo trường... Tầng này sẽ gọi các phương thức từ tầng *DataAccess*. Để gọi được từ tầng này, phải sử dụng chức năng thêm các tập tin *.dll vào dự án (Click phải lên References, chọn Add Reference..., chọn *DataAccess*, nhấn OK), sau đó sử dụng lệnh using để gọi thư viện như sau trên mỗi lớp:

```
using DataAccess;
```

7.4.1. Lớp *CategoryBL*

Lớp *CategoryBL* là lớp dùng để thực hiện các chức năng của bảng *Category*, các chức năng chủ yếu của bảng này bao gồm các phương thức cơ bản như: lấy hết, thêm, xoá, sửa... Khi cần thêm các chức năng khác, có thể viết thêm. Chương trình 7.9 minh họa các phương thức của lớp *CategoryBL*.

Chương trình 7.9. Các phương thức cơ bản của lớp CategoryBL

```
// Lớp CategoryBL có các phương thức xử lý bảng Category
public class CategoryBL
{
    //Đối tượng CategoryDA từ DataAccess
    CategoryDA categoryDA = new CategoryDA();
    //Phương thức lấy hết dữ liệu
    public List<Category> GetAll()
    {
```

```
        return categoryDA.GetAll();
    }
    //Phương thức thêm dữ liệu
    public int Insert(Category category)
    {
        return categoryDA.Insert_Update_Delete(category, 0);
    }
    //Phương thức cập nhật dữ liệu
    public int Update(Category category)
    {
        return categoryDA.Insert_Update_Delete(category, 1);
    }
    //Phương thức xoá dữ liệu truyền vào ID
    public int Delete(Category category)
    {
        return categoryDA.Insert_Update_Delete(category, 2);
    }
}
```

7.4.2. Lớp FoodBL

Lớp FoodBL dùng để thực hiện một số chức năng của bảng Food, các chức năng chủ yếu của bảng này bao gồm các phương thức cơ bản như: lấy hết, thêm, xoá, sửa. Ngoài ra, có thể viết thêm các hàm khác như tìm kiếm, lấy theo khoá ngoại... Chương trình 7.10 minh họa các phương thức của lớp FoodBL.

Chương trình 7.10. Các phương thức cơ bản của lớp FoodBL

```
// Lớp FoodBL có các phương thức xử lý bảng Food
public class FoodBL
{
    //Đối tượng CategoryDA từ DataAccess
    FoodDA foodDA = new FoodDA();
    //Phương thức lấy hết dữ liệu
    public List<Food> GetAll()
    {
        return foodDA.GetAll();
    }
    // Phương thức lấy về đối tượng Food theo khoá chính
    public Food GetByID(int ID)
    {
        // Lấy hết
        List<Food> list = GetAll();
        // Duyệt để tìm kiếm
        foreach (var item in list)
        {
            if (item.ID == ID) // Nếu gặp khoá chính
                return item; // thì trả về kết quả
        }
        return null;
    }
    //Phương thức tìm kiếm theo khoá
    public List<Food> Find(string key)
    {
```

```
List<Food> list = GetAll(); // Lấy hết
List<Food> result = new List<Food>();
// Duyệt theo danh sách
foreach (var item in list)
{
    // Nếu từng trường chứa từ khoá
    if (item.ID.ToString().Contains(key)
        || item.Name.Contains(key)
        || item.Unit.Contains(key)
        || item.Price.ToString().Contains(key)
        || item.Notes.Contains(key))
        result.Add(item); // Thì thêm vào danh sách kết quả
}
return result;
}
//Phương thức thêm dữ liệu
public int Insert(Food food)
{
    return foodDA.Insert_Update_Delete(food, 0);
}
//Phương thức cập nhật dữ liệu
public int Update(Food food)
{
    return foodDA.Insert_Update_Delete(food, 1);
}
//Phương thức xoá dữ liệu với ID cho trước
public int Delete(Food food)
{
    return foodDA.Insert_Update_Delete(food, 2);
}
```

Lớp này nếu sau này trong quá trình xây dựng ứng dụng, có nhu cầu có thể viết thêm các phương thức khác bổ sung. Tương tự như tầng DataAccess, khi viết xong tầng BusinessLogic, có thể kiểm tra lỗi và tạo tập tin *.dll bằng cách click phải lên BusinessLogic, chọn Build, khi đó nếu có lỗi sai, hệ thống sẽ báo để sửa lỗi trước khi viết qua tầng mới.

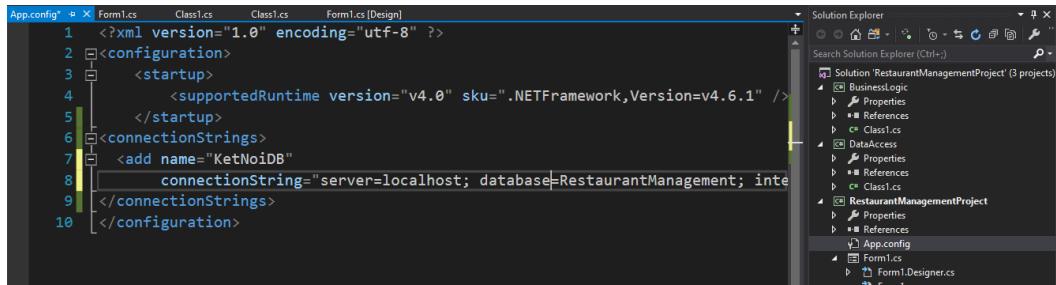
7.5. Lớp giao diện người dùng (User Interface – UI)

7.5.1. Tạo chuỗi kết nối

Chuỗi kết nối được tạo ra bằng cách thêm đoạn mã vào trong tập tin App.config của dự án: Trong Solution Explorer, nhấp đúp chuột vào tập tin App.config, thêm đoạn như sau vào trong thẻ configuration (Hình 7.8):

```
<connectionStrings>
    <add name="ConnectionStringName"
        connectionString="server=localhost; database=RestaurantManagement;
        integrated security=true;" />
</connectionStrings>
```

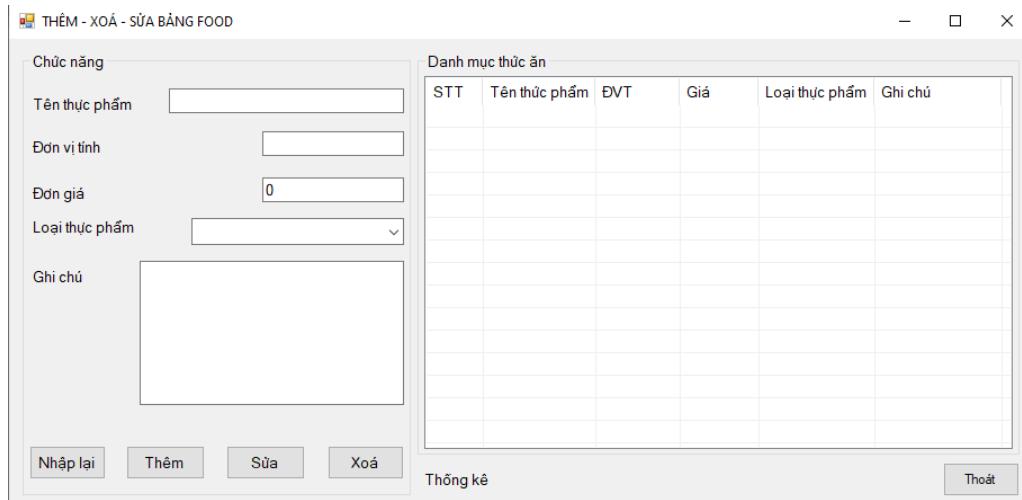
Lưu ý: Giá trị “ConnectionStringName” là tên của chuỗi kết nối, các giá trị được in đậm là các giá trị thay đổi theo tên của Server và tên cơ sở dữ liệu trong SQL Server. Ngoài ra, chuỗi kết nối có thể thêm vào lệnh có chứa tên tài khoản và mật khẩu để tăng tính bảo mật.



Hình 7.9. Cấu hình chuỗi kết nối trong tập tin App.config

7.5.2. Thiết kế giao diện chương trình

Giao diện chương trình minh họa ở đây dùng để quản lý bảng Food, bao gồm các chức năng chính như: Thêm, xoá, sửa, tìm kiếm... Giao diện chương trình được thiết kế như Hình 7.10, chi tiết các thành phần trên giao diện được mô tả như trong Bảng 7.3.



Hình 7.10. Giao diện chức năng quản lý bảng Food

Để hiển thị thông tin, có thể sử dụng ListView hoặc DataGridView, ở đây sử dụng ListView. Dữ liệu Loại thực phẩm được đọc từ bảng Category và đưa vào ComboBox.

Bảng 7.3. Các thành phần trên giao diện

Thành phần	Tên thành phần	Thuộc tính
Form	frmFood	Text= THÊM - XOÁ - SỬA BẢNG FOOD
GroupBox	grpLeft	Text = Chức năng; Anchor = Top, Bottom, Left
GroupBox	grpRight	Text = Danh mục thực ăn; Anchor = Top, Bottom, Left, Right

Bảng 7.3. Các thành phần trên giao diện (tiếp)

Thành phần	Tên thành phần	Thuộc tính
Label	Tên mặc định	Text là tiêu đề cho các thành phần như “Tên thực phẩm”, “Đơn vị tính”, “Đơn giá”, “Loại thực phẩm”, “Ghi chú”
TextBox	txtName	Anchor= Top, Left
	txtUnit	
	txtPrice	
ComboBox	cbbCategory	Anchor= Top, Left
TextBox	txtNotes	Anchor= Top, Left; Multiline = True
Button	cmdClear	Anchor= Bottom;
	cmdAdd	Text là tiêu đề cho các nút bấm như “Nhập lại”, “Thêm”,
	cmdUpdate	“Sửa”, “Xoá”.
	cmdDelete	
ListView	lsvFood	Anchor= Top, Bottom, Left, Right; Columns={STT, Tên thực phẩm, ĐVT, Giá, Loại thực phẩm, Ghi chú}; FullRowSelect= true; GridLines= true; MultiSelect= false; View= Details
Label	lblStatistic	Text= Thống kê
Button	cmdExit	Text= Thoát; Anchor= Bottom, Right

Các thành phần trên giao diện nếu có cùng kiểu thì nên để thuộc tính Name có cùng tiền tố. Thuộc tính Anchor nên thiết lập phù hợp để phòng trường hợp người dùng phóng to hay thu nhỏ màn hình. Các thành phần mà gọi trong phần mã nguồn thì mới cần đặt tên, các thành phần còn lại không cần đặt tên mà để tên mặc định.

Sau khi thiết kế xong giao diện, nút Thoát và nút Nhập lại được xử lý bằng cách nhấp đôi chuột lên nút để tạo mã nguồn tự động sự kiện *OnClick* như trong Chương trình 7.11.

Chương trình 7.11. Sự kiện khi Click nút Thoát và nút Nhập lại

```

/// <summary>
/// Sự kiện click nút Thoát
/// </summary>
private void cmdExit_Click(object sender, EventArgs e)
{
    // Gọi phương thức Exit() để thoát ứng dụng
    Application.Exit();
}
/// <summary>
/// Sự kiện khi click nút Nhập lại

```

```
/// </summary>
private void cmdClear_Click(object sender, EventArgs e)
{
    //Gán các ô bằng giá trị mặc định
    txtName.Text = "";
    txtPrice.Text = "0";
    txtUnit.Text = "";
    txtNotes.Text = "";
    // Thiết lập index = 0 cho ComboBox
    if(cbbCategory.Items.Count >0)
        cbbCategory.SelectedIndex = 0;
}
```

Sau khi viết lệnh cho các nút này, nhấn F5 và chạy thử chương trình, đồng thời nhấn nút để kiểm tra các sự kiện đã viết. Các mục tiếp theo sẽ đề cập đến các chức năng còn lại.

7.5.3. Các phương thức tự động tải dữ liệu

Để thực hiện được các thao tác với giao diện đã thiết kế, cần phải xây dựng một số phương thức để tải dữ liệu tự động. Để gọi các đối tượng từ tầng DataAccess và BusinessLogic, phải sử dụng lệnh using như sau (sau khi đã tham chiếu vào Reference):

```
using BusinessLogic;
using DataAccess;
```

Phương thức đầu tiên cần phải xây dựng là phương thức dùng để tải dữ liệu từ bảng Category, đưa vào một danh sách, sau đó đổ vào ComboBox. Phương thức thứ hai cũng lấy dữ liệu từ bảng Food, đưa vào một danh sách, sau đó đổ vào ListView. Phương thức này lần lượt lấy dữ liệu từ các trường trong bảng Food và gắn vào từng cột trong ListView, riêng trường CategoryID, dùng LINQ để đọc và hiển thị tên thay vì hiển thị theo những chữ số (ID). Để thuận tiện trong khi xử lý sau này, các danh sách chứa thông tin về dữ liệu hai bảng này được khai báo toàn cục trong lớp Form, đồng thời khai báo một đối tượng Food biểu diễn giá trị đang chọn hiện hành.

```
// Danh sách toàn cục bảng Category
List<Category> listCategory = new List<Category>();
// Danh sách toàn cục bảng Food
List<Food> listFood = new List<Food>();
// Đối tượng Food đang chọn hiện hành
Food foodCurrent = new Food();
```

Hai phương thức trên được gọi trong sự kiện *Form_Load* nhằm mục đích dữ liệu được tải lên khi chạy chương trình. Hai phương thức và sự kiện *Form_Load* được giải thích theo các dòng lệnh trong Chương trình 7.12.

Chương trình 7.12. Hai phương thức tải dữ liệu và sự kiện Form_Load

```
private void Form1_Load(object sender, EventArgs e)
{
    //Đổ dữ liệu vào ComboBox
```

```
LoadCategory();
// Đổ dữ liệu vào ListView
LoadFoodDataToListView();
}
/// <summary>
/// Phương thức tải dữ liệu từ bảng Category vào Combobox
/// </summary>
private void LoadCategory()
{
    //Gọi đối tượng CategoryBL từ tầng BusinessLogic
    CategoryBL categoryBL = new CategoryBL();
    // Lấy dữ liệu gán cho biến toàn cục listCategory
    listCategory = categoryBL.GetAll();
    // Chuyển vào Combobox với dữ liệu là ID, hiển thị là Name
    cbbCategory.DataSource = listCategory;
    cbbCategory.ValueMember = "ID";
    cbbCategory.DisplayMember = "Name";
}
/// <summary>
/// Phương thức tải dữ liệu từ bảng Food vào ListView
/// </summary>
public void LoadFoodDataToListView()
{
    //Gọi đối tượng FoodBL từ tầng BusinessLogic
    FoodBL foodBL = new FoodBL();
    // Lấy dữ liệu
    listFood= foodBL.GetAll();
    int count = 1; // Biến số thứ tự
    // Xoá dữ liệu trong ListView
    lsvFood.Items.Clear();
    // Duyệt mảng dữ liệu để đưa vào ListView
    foreach (var food in listFood)
    {
        // Số thứ tự
        ListViewItem item = lsvFood.Items.Add(count.ToString());
        // Đưa dữ liệu Name, Unit, price vào cột tiếp theo
        item.SubItems.Add(food.Name);
        item.SubItems.Add(food.Unit);
        item.SubItems.Add(food.Price.ToString());
        // Theo dữ liệu của bảng Category ID, lấy Name để hiển thị
        string foodName = listCategory
            .Find(x => x.ID == food.FoodCategoryID).Name;
        item.SubItems.Add(foodName);
        // Đưa dữ liệu Notes vào cột cuối
        item.SubItems.Add(food.Notes);
        count++;
    }
}
```

Phương thức thứ ba là sự kiện khi người dùng click chuột vào một dòng trên *ListView*, dữ liệu sẽ được lấy ra và gán cho biến của đối tượng *Food* hiện hành, đồng thời giá trị của các trường được đưa vào các ô bên trái. Do đã thiết lập từ trước các thuộc

tính của *ListView* là *FullRowSelect= true* và *MultiSelect= false* (xem Bảng 7.3) nên khi người dùng nhấp chuột vào vùng dữ liệu thì chỉ 1 dòng được chọn và dòng đó phải chọn cả dòng. Chi tiết sự kiện này được giải thích trong Chương trình 7.13.

Chương trình 7.13. Sự kiện click lên dòng của ListView

```
private void lsvFood_Click(object sender, EventArgs e)
{
    // Duyệt toàn bộ dữ liệu trong ListView
    for (int i = 0; i < lsvFood.Items.Count; i++)
    {
        // Nếu có dòng được chọn thì lấy dòng đó
        if (lsvFood.Items[i].Selected)
        {
            // Lấy các tham số và gán dữ liệu vào các ô
            foodCurrent = listFood[i];
            txtName.Text = foodCurrent.Name;
            txtUnit.Text = foodCurrent.Unit;
            txtPrice.Text = foodCurrent.Price.ToString();
            txtNotes.Text = foodCurrent.Notes;
            // Lấy index của Combobox theo FoodCategoryID
            cbbCategory.SelectedIndex = listCategory
                .FindIndex(x => x.ID == foodCurrent.FoodCategoryID);
        }
    }
}
```

Sau khi viết lệnh, nhấn F5 và chạy thử chương trình để xem dữ liệu được hiển thị trong *ListView* và *ComboBox*, nếu bị lỗi có thể xem lại chuỗi kết nối và các tham số trong *App.config*. Click chọn dòng trong *ListView* để dữ liệu được đưa vào trong các ô như trong Hình 7.11.

STT	Tên thức phẩm	ĐVT	Giá	Loại thực phẩm	Ghi chú
1	Rau muống x...	Đĩa	20000	Rau	
2	Cơm chiên D...	Đĩa nhỏ	35000	Cơm	3 người ăn
3	Cơm chiên D...	Đĩa lớn	40000	Cơm	4 người ăn
4	Éch thuỷ rôm	Đĩa	70000	Hải sản	
5	Sò lông nướng...	Đĩa	80000	Hải sản	
6	Càng cua hấp	Đĩa	100000	Hải sản	
7	Canh cải	Tô	20000	Canh	
8	Gà nướng mu...	Con	180000	Gà	
9	Bia 333	Chai	12000	Bia	
10	Bia Heniken	Chai	20000	Bia	
11	Súp cua	Tô	15000	Khai vị	
12	Thịt kho dưa	Đĩa	25000	Thịt	Theo thời giá
13	Thịt kho hành...	Đĩa	50000	Khai vị	

Hình 7.11. Chương trình khi hoàn thành các phương thức lấy dữ liệu

7.5.4. Xử lý nút thêm, xoá, sửa

Nút Thêm được xử lý bằng cách xây dựng một phương thức cho phép chèn dữ liệu vào bảng Food dựa trên các dữ liệu nhập. Sau khi kiểm tra các dữ liệu nhập từ người dùng, hệ thống sẽ lấy các tham số để truyền cho đối tượng Food, đối tượng FoodBL sẽ gọi phương thức Insert và trả về giá trị kiểu int. Trong bảng Food này, vì ID là tự tăng nên giá trị nhận vào được gán mặc định (bằng 0), FoodCategoryID được lấy từ ComboBox. Sau khi chèn dữ liệu sẽ có hộp thoại thông báo cho người dùng, đồng thời tải lại dữ liệu vào ListView bằng phương thức LoadFoodDataToListView(). Chi tiết phương thức thêm dữ liệu và sự kiện khi nhấn nút Thêm được mô tả như trong Chương trình 7.14.

Chương trình 7.14. Sự kiện khi nhấn nút Thêm và phương thức thêm dữ liệu

```
/// <summary>
/// Sự kiện khi click nút Thêm
/// </summary>
private void cmdAdd_Click(object sender, EventArgs e)
{
    // Gọi phương thức thêm dữ liệu
    int result = InsertFood();
    if(result > 0) // Nếu thêm thành công
    {
        // Thông báo kết quả
        MessageBox.Show("Thêm dữ liệu thành công");
        // Tải lại dữ liệu cho ListView
        LoadFoodDataToListView();
    }
    // Nếu thêm không thành công thì thông báo cho người dùng
    else MessageBox.Show("Thêm dữ liệu không thành công. Vui lòng kiểm tra
lại dữ liệu nhập");
}

/// <summary>
/// Phương thức thêm dữ liệu cho bảng Food
/// </summary>
/// <returns>Trả về số dương nếu thành công, ngược lại trả về số
âm</returns>
public int InsertFood()
{
    //Khai báo đối tượng Food từ tầng DataAccess
    Food food = new Food();
    food.ID = 0;
    // Kiểm tra nếu các ô nhập khác rỗng
    if (txtName.Text == "" || txtUnit.Text == "" || txtPrice.Text == "")
        MessageBox.Show("Chưa nhập dữ liệu cho các ô, vui lòng nhập lại");
    else {
        //Nhận giá trị Name, Unit, và Notes từ người dùng nhập vào
        food.Name = txtName.Text;
        food.Unit = txtUnit.Text;
        food.Notes = txtNotes.Text;
```

```
// Giá trị price là giá trị số nên cần bắt lỗi khi người dùng nhập sai
int price = 0;
try
{
    // Cố gắng lấy giá trị
    price = int.Parse(txtPrice.Text);
}
catch
{
    // Nếu sai, gán giá = 0
    price = 0;
}
food.Price = price;
// Giá trị FoodCategoryID được lấy từ ComboBox
food.FoodCategoryID = int.Parse(cbbCategory.SelectedValue.
ToString());
// Khao báo đối tượng FoodBL từ tầng Business
FoodBL foodBL = new FoodBL();
// Chèn dữ liệu vào bảng
return foodBL.Insert(food);
}
return -1;
}
```

Để xử lý nút Xoá và nút Sửa, cần dựa trên biến foodCurrent đã khai báo trong Mục 7.3.5, biến này sẽ lưu đối tượng đang được chọn và ta cần xoá hoặc cập nhật đối tượng này. Khi nhấn vào nút Xoá, người dùng sẽ được hỏi lại một câu để khẳng định có chắc chắn là xoá hay không (tránh trường hợp click nhầm), sau khi khẳng định chắc chắn, hệ thống sẽ lấy đối tượng Food hiện hành để xoá. Sau khi xoá, hệ thống sẽ có hộp thoại thông báo cho người dùng, đồng thời tải lại bảng phương thức LoadFoodDataToListView(). Chi tiết sự kiện khi nhấn nút Xoá được mô tả như trong Chương trình 7.15.

Chương trình 7.15. Xử lý sự kiện nút Xoá

```
/// <summary>
/// Sự kiện khi click chuột lên nút Xoá
/// </summary>
private void cmdDelete_Click(object sender, EventArgs e)
{
    // Hỏi người dùng có chắc chắn xoá hay không? Nếu đồng ý thì
    if(MessageBox.Show("Bạn có chắc chắn muốn xoá mẫu tin này?", "Thông
báo",
        MessageBoxButtons.YesNo, MessageBoxIcon.Warning) ==
DialogResult.Yes)
    {
        // Khai báo đối tượng FoodBL từ BusinessLogic
        FoodBL foodBL = new FoodBL();
        if (foodBL.Delete(foodCurrent) > 0)// Nếu xoá thành công
        {
            MessageBox.Show("Xoá thực phẩm thành công");
            // Tải dữ liệu lên ListView
        }
    }
}
```

```
        LoadFoodDataToListView();
    }
    else MessageBox.Show("Xoá không thành công");
}
}
```

Khi nhấn vào nút Sửa, hệ thống sẽ lấy giá trị hiện hành (biến foodCurrent) làm giá trị để sửa, ID được giữ lại, các giá trị còn lại được cập nhật bằng cách dựa trên dữ liệu mà người dùng nhập vào. Sau khi kiểm tra tính đúng đắn của dữ liệu nhập, đối tượng FoodBL sẽ gọi phương thức Update và trả về giá trị kiểu int, giá trị FoodCategoryID được lấy từ ComboBox. Sau khi cập nhật dữ liệu sẽ có hộp thoại thông báo cho người dùng, đồng thời tải lại dữ liệu vào ListView bằng phương thức LoadFoodDataToListView(). Chi tiết phương thức cập nhật dữ liệu và sự kiện khi nhấn nút sửa được mô tả như trong Chương trình 7.16.

Chương trình 7.16. Sự kiện khi nhấn nút Sửa và phương thức cập nhật dữ liệu

```
/// <summary>
/// Sửa kiện khi nhấn nút Sửa
/// </summary>
private void cmdUpdate_Click(object sender, EventArgs e)
{
    // Gọi phương thức cập nhật dữ liệu
    int result = UpdateFood();
    if (result > 0) // Nếu cập nhật thành công
    {
        // Thông báo kết quả
        MessageBox.Show("Cập nhật dữ liệu thành công");
        // Tải lại dữ liệu cho ListView
        LoadFoodDataToListView();
    }
    // Nếu thêm không thành công thì thông báo cho người dùng
    else MessageBox.Show("Cập nhật dữ liệu không thành công. Vui lòng kiểm
tra lại dữ liệu nhập");
}

/// <summary>
/// Phương thức cập nhật dữ liệu cho bảng Food
/// </summary>
/// <returns>Trả về số dương nếu cập nhật thành công, ngược lại là số
âm</returns>
public int UpdateFood()
{
    //Khai báo đối tượng Food và lấy đối tượng hiện hành
    Food food = foodCurrent;
    // Kiểm tra nếu các ô nhập khác rỗng
    if (txtName.Text == "" || txtUnit.Text == "" || txtPrice.Text == "")
        MessageBox.Show("Chưa nhập dữ liệu cho các ô, vui lòng nhập lại");
    else
    {
        //Nhận giá trị Name, Unit, và Notes khi người dùng sửa
        food.Name = txtName.Text;
        food.Unit = txtUnit.Text;
```

```
    food.Notes = txtNotes.Text;
    // Giá trị price là giá trị số nên cần bắt lỗi khi người dùng nhập sai
    int price = 0;
    try
    {
        // Chuyển giá trị từ kiểu văn bản qua kiểu int
        price = int.Parse(txtPrice.Text);
    }
    catch
    {
        // Nếu sai, gán giá = 0
        price = 0;
    }
    food.Price = price;
    // Giá trị FoodCategoryID được lấy từ ComboBox
    food.FoodCategoryID =
int.Parse(cbbCategory.SelectedValue.ToString());
    // Khao báo đối tượng FoodBL từ tầng Business
    FoodBL foodBL = new FoodBL();
    // Cập nhật dữ liệu trong bảng
    return foodBL.Update(food);
}
return -1;
}
```

Sau khi viết lệnh cho các nút, nhấn F5 và chạy thử chương trình. Kiểm tra thử khi nhấn các nút Thêm, Xoá, Sửa. Nhập sai dữ liệu để kiểm tra các hộp thoại.

7.6. Kết chương

Chương này giới thiệu mô hình đa tầng, kiến trúc của mô hình đa tầng, đồng thời giới thiệu minh họa mô hình đa tầng với ba tầng là tầng Giao diện, tầng Xử lý logic và tầng Truy xuất dữ liệu trên Visual Studio và SQL Server. Thông qua ví dụ minh họa, người đọc có thể hiểu được phần nào việc xây dựng mô hình đa tầng và việc gọi các phương thức, thuộc tính giữa các tầng. Ở đây, để đơn giản và dễ hiểu, chỉ xây dựng mô hình đa tầng với việc truy xuất và xử lý dữ liệu trên bảng Food và bảng Category. Thông qua đó, người đọc sau khi thực hiện được sẽ tự phát triển và xây dựng các chức năng khác để hoàn thiện chương trình và phát triển được nhiều Project trên một Solution.

Bài tập:

1. Từ các bước minh họa ở trên, xây dựng chương trình mô hình ba tầng cho bảng Category và bảng Food.
2. Xây dựng chương trình cho phép nhập liệu tất cả các bảng (Xem cơ sở dữ liệu phần Phụ lục).
3. Xây dựng phần phân quyền, cho phép gán các quyền như: Quản lý, Kế toán, Nhân viên, Admin.
4. Xây dựng chương trình quản lý nhà hàng hoàn chỉnh với các chức năng như: Đặt món, tách bàn, thanh toán...

TÀI LIỆU THAM KHẢO

Nguyễn Minh Hiệp, Nguyễn Văn Phúc. (2009). Giáo trình lập trình cơ sở dữ liệu. Lâm Đồng, Việt Nam: Trường Đại học Đà Lạt.

Jason Price. (2004). Mastering C# Database Programming. Retrieved from
<https://epdf.pub/mastering-c-database-programming55279.html>

[https://medium.com/@hidayatarg/n-tier-layer-architecture-in-c-15b8fe97283c#:~:text=N%20represent%20a%20number%20and,\(Layers\)%20of%20an%20application.&text=It%20comprise%20of%203%2Dtiers,ands%20Layers%20are%20used%20interchangeably](https://medium.com/@hidayatarg/n-tier-layer-architecture-in-c-15b8fe97283c#:~:text=N%20represent%20a%20number%20and,(Layers)%20of%20an%20application.&text=It%20comprise%20of%203%2Dtiers,ands%20Layers%20are%20used%20interchangeably)

<https://www.connectionstrings.com>

<https://www.entityframeworktutorial.net/>

<https://docs.microsoft.com/en-us/ef/ef6/>

<https://www.entityframeworktutorial.net/>

PHỤ LỤC:

CƠ SỞ DỮ LIỆU QUẢN LÝ NHÀ HÀNG (ĂN UỐNG)

1. Tên các bảng

STT	Tên trường	Kiểu	Null	Ghi chú
-----	------------	------	------	---------

Bảng Table				
1	ID	int		Tự tăng; Khóa chính
2	Name	nvarchar(1000)	x	Tên bàn: Ví dụ: Bàn VIP, Bàn Khách đặc biệt.
3	Status	int		Trạng thái: 0: chưa đặt; 1: Đã đặt; 2: Có khách
4	Capacity	int	x	Số chỗ ngồi tối đa

Ghi chú: Bảng này lưu thông tin về bàn ăn. Một bàn sẽ có tên, bàn chứa tối đa bao nhiêu chỗ ngồi.

Bảng Category				
1	ID	int		Tự tăng; Khóa chính
2	Name	nvarchar(1000)		Tên đồ ăn, thức uống, mặc định “Chưa đặt tên”
3	Type	int		Đồ ăn (1) hay thức uống (2)

Ghi chú: Bảng này lưu thông tin về loại đồ ăn, thức uống. Nếu Type = 1 là thức ăn; nếu Type = 2 là đồ uống.

Bảng Food				
1	ID	int		Tự tăng; Khóa chính
2	Name	nvarchar(1000)		Tên đồ ăn, mặc định “Chưa đặt tên”
3	Unit	nvarchar(100)		Đơn vị tính
4	FoodCategoryID	int		Thuộc loại đồ ăn, thức uống nào
5	Price	int		Giá, mặc định 0
6	Notes	Nvarchar(3000)	X	Ghi chú về món ăn

Ghi chú: Bảng này lưu thông tin về món ăn, món uống và giá của từng món.

Bảng Bills				
1	ID	int		Tự tăng; Khóa chính
2	Name	nvarchar(1000)		Tên hóa đơn, mặc định “Hóa đơn thanh toán”
3	TableID	Int		Ngồi bàn nào
4	Amount	Int		Tổng tiền phải thanh toán, mặc định 0
5	Discount	Float	X	Số tiền giảm giá, mặc định 0
6	Tax	float	X	Thuế, mặc định 0
7	Status	bit		Đã thanh toán hay chưa (1 hoặc 0), mặc định 0
8	Account	nvarchar(100)		Tài khoản nào đang đăng nhập
9	CheckoutDate	smalldatetime		Ngày in phiếu, mặc định ngày hiện tại

Ghi chú: Bảng này lưu thông tin về phiếu thu tiền; Biên lai chỉ lưu các thông tin cơ bản, chi tiết món ăn sẽ có trong bảng BillDetails.

Giáo trình Lập trình Cơ sở dữ liệu

Bảng BillDetails				
1	ID	int		Tự tăng; Khóa chính
2	InvoicelD	int		Thuộc hóa đơn nào
3	FoodID	int		Đồ ăn, thức uống nào
4	Quantity	int		Số lượng, mặc định 0

Ghi chú: Bảng này lưu thông tin về các món ăn và số lượng trong phiếu;

Bảng Account				
1	AccountName	nvarchar(100)		Tên tài khoản; Khóa chính
2	Password	nvarchar(200)		Mật khẩu
3	FullName	nvarchar(1000)		Họ tên
4	Email	nvarchar(1000)	X	
5	Tell	nvarchar(200)		
6	DateCreated	smalldatetime		Ngày tạo tài khoản, mặc định là ngày hiện tại

Ghi chú: Bảng này lưu thông tin về các tài khoản đăng nhập; Thông tin tài khoản sẽ kèm theo Họ và tên, cũng như các thông tin cơ bản khác.

Bảng Role				
1	ID	int		Tự tăng, khóa chính
2	RoleName	nvarchar(1000)		Tên quyền
3	Path	nvarchar(3000)	X	Đường dẫn (hoặc tên Control)
4	Notes		X	

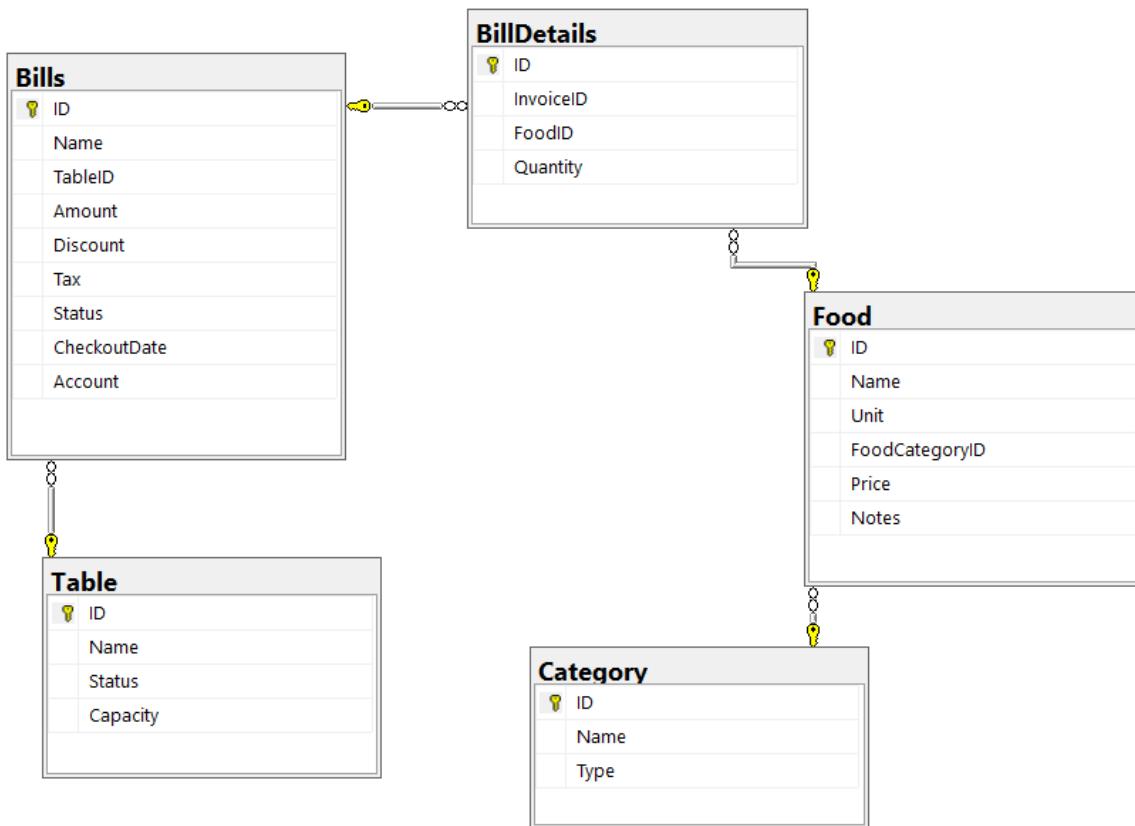
Ghi chú: Bảng này lưu thông tin về các quyền trong chương trình. Mỗi quyền sẽ có một tên quyền: Ví dụ: Quản lý, Kế toán, Nhân viên, ... Path lưu trữ thông tin về Control hoặc đường dẫn chứa quyền.

Bảng RoleAccount				
1	AccountName	nvarchar(100)		Khóa chính
2	RoleID	Int		Khóa chính
3	Actived	bit		Có kích hoạt hay không
4	Notes	nvarchar(3000)	X	Ghi chú

Ghi chú: Bảng này dùng để phân quyền. Mỗi tài khoản được gắn với 1 quyền và có được kích hoạt hay không.

2. Sơ đồ quan hệ

-) Sơ đồ quan hệ các bảng chức năng



-) Sơ đồ quan hệ các bảng phân quyền

