第 6 章 使用 ADO.NET 进行数据库编程开发

6.1 数据库设计基础及 SQL 语句演练

6.1.3 网络点餐管理系统数据库设计

数据库设计(Database Design)是指对于一个给定的应用环境，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求（信息要求和处理要求）。在数据库领域内，常常把使用数据库的各类系统统称为数据库应用系统。

数据库设计的设计内容包括：需求分析、概念设计、逻辑设计、物理设计、数据库的实施和数据库的运行和维护等。

1. 需求分析

假定一个餐厅的每张餐台都配备了一台无线联网的触屏式终端，需要开发一个具有网络点餐功能的管理系统，将顾客在终端点餐的数据自动传送到服务器。在结账时，收银台可以通过该系统自动计算出每张餐台顾客的消费金额并打印消费清单，整个系统构成的示意图如图 6.3 所示。

根据系统功能需求，从顾客的角度来分析可知，顾客要点餐，必须有菜单(菜品列表)，考虑的餐厅菜品的多样性，人为的将菜品按照菜系分类，从而得到菜系和菜品两个实体。

而顾客是流动的，一般餐厅不关心具体顾客的信息，而是根据哪张餐台、点了哪些菜品项进行结账，因此引入餐台和餐台对应的点菜单两个实体。一张餐台的点菜单有多个菜品项，为了避免数据冗余，可以建立顾客点菜的明细表(消费清单)。

服务员根据餐台进行结账，这里建立了餐台、餐台点菜单和点菜明细之间的关系，从而得出结账单。

站在餐厅商家的角度分析，老板可能要在某个时间段进行菜品分析，统计得到最受欢迎菜品，或进行营业额统计，通过结账单，可以检索到餐台点餐单及明细等信息。

有了上面的分析，我们可以从实体着手，找出每个实体及其属性，再根据实体间的联系，做出局部 E-R 图，最后组合成整体 E-R 图。E-R 图有助于理清实体之间的关系，进行合理的逻辑设计，最终得到系统所需建立的数据表。

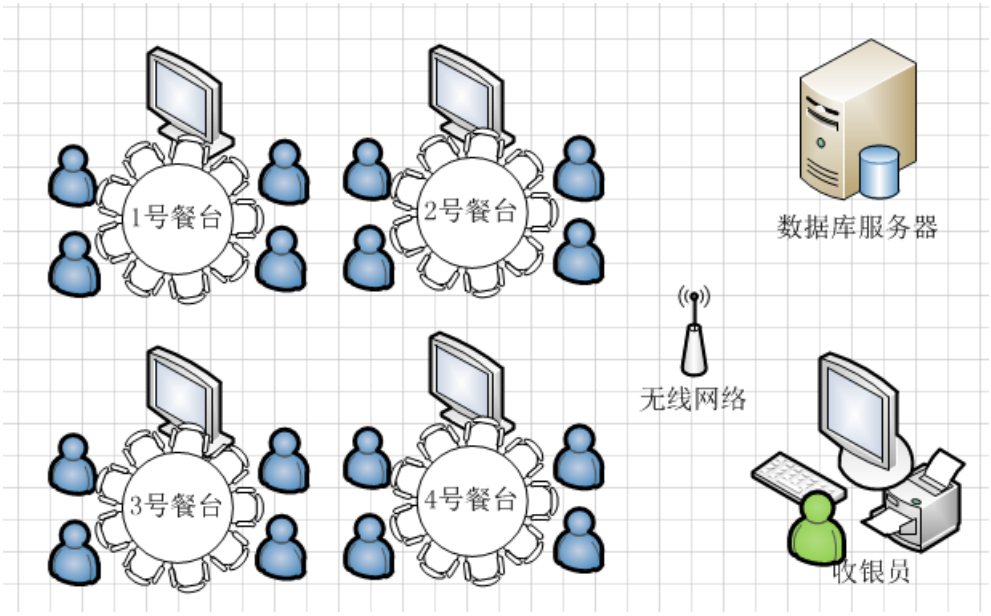

图 6.3　网络点餐管理系统示意图

2. 概念设计

根据需求分析，画出每个实体及其属性图，找出实体间的联系，并得出其关系模式。图 6.4~6.9 为各个实体及其属性以及局部 E-R 图，图 6.10 为整体 E-R 图。注意，E-R 图并非一次成型的，而是根据对需求的理解，经过反复修改最终得到的。

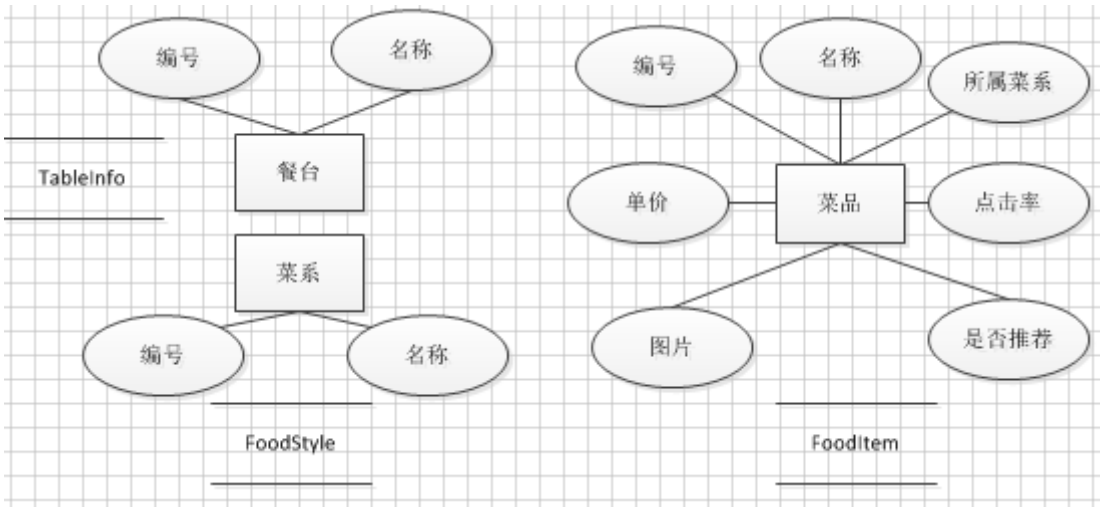

图 6.4　餐台(TableInfo)、菜系(FoodStyle)和菜品(FoodItem)实体图

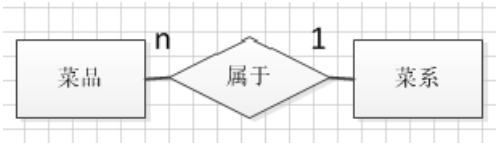

图 6.5　菜系与菜品的局部 E-R 图

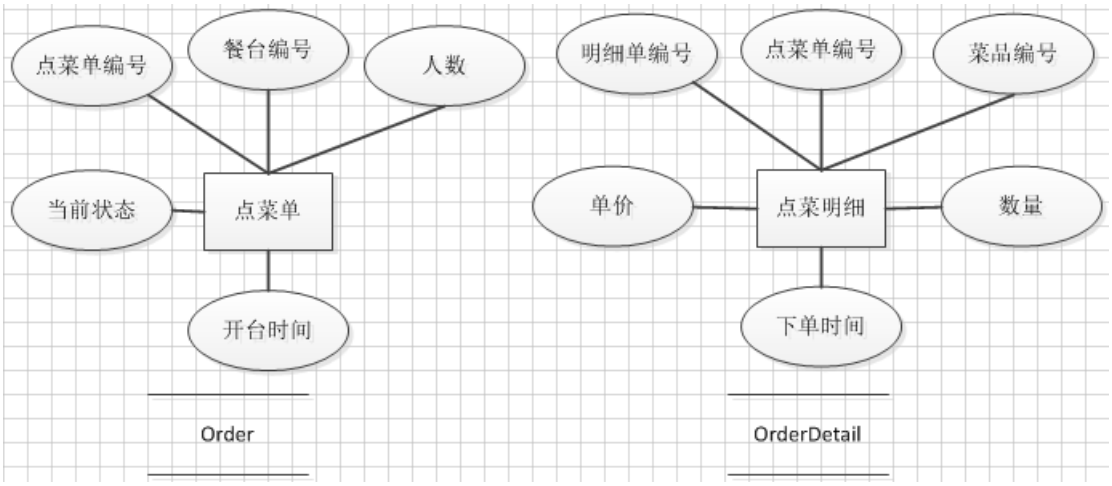

图 6.6　点菜单(Order)、点菜明细(OrderDetail)的实体图

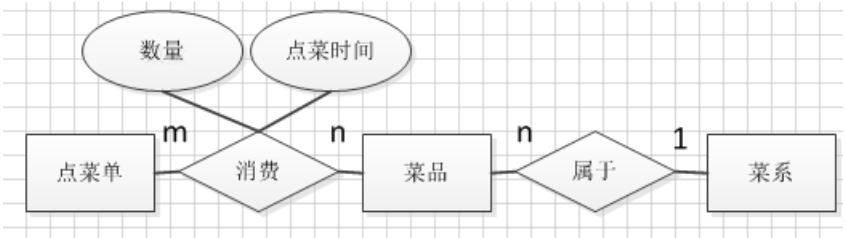

图 6.7　点菜单(Order)、菜品(FoodStyle)和菜系(FoodItem)的局部 E-R 图

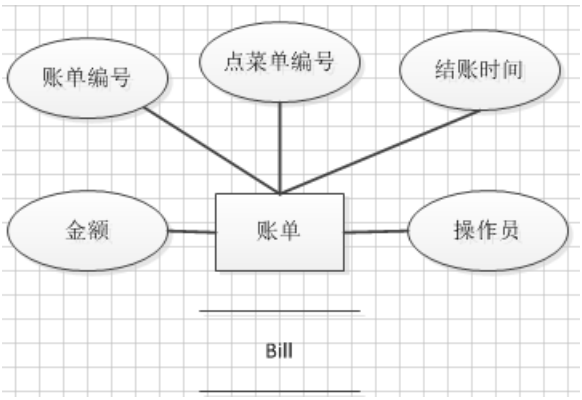

图 6.8　账单实体图

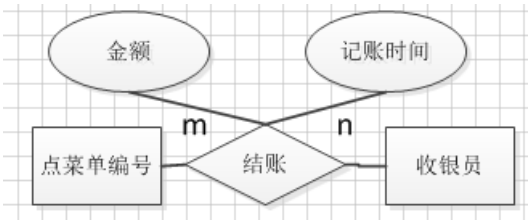

图 6.9　餐台与收银员的局部 E-R 图

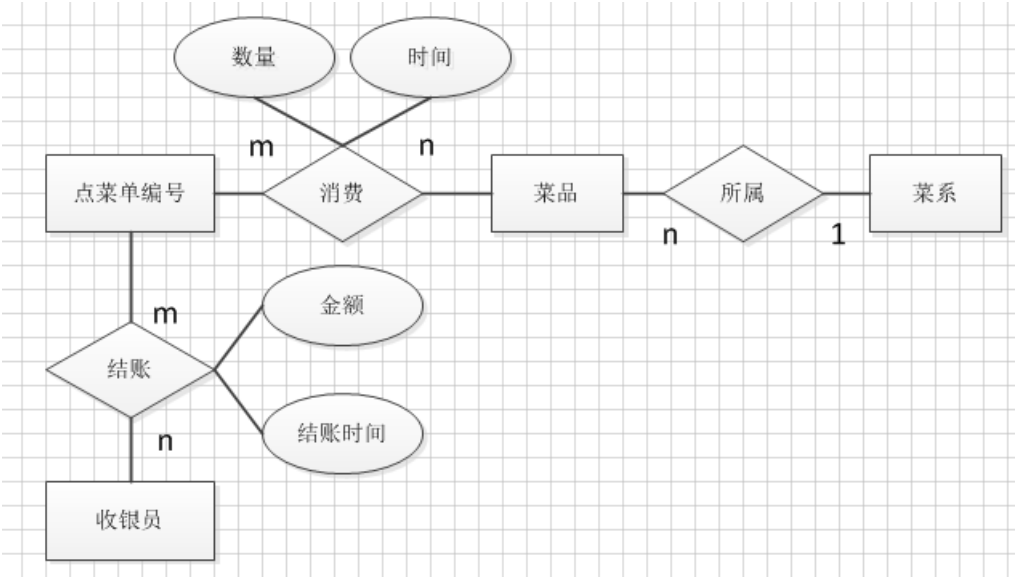

图 6.10　整体 E-R 图

3. 逻辑设计

根据 E-R 图得到数据库的关系模型如下

(1) 基础信息：餐台、操作员。

① 餐台信息(餐台编号,类型)。

② 操作员(操作员编号，用户名，密码，性别，联系电话)。

(2) 菜品与菜系的关系不需要单独建立一个表，而是将菜系编号作为菜品的外键，得到其关系模型如下：

① 菜系(菜系编号，菜系名称)。

② 菜品(菜品编号,菜系编号,菜品名称,单价，图片位置,推荐,单击率)。

(3) 每张餐台都有唯一一张消费单据，作为结账的依据，根据其与菜品的联系，得到消费明细单的关系模型如下：

① 消费单（点餐单编号,餐台编号,人数，开台时间，当前状态）。

② 消费明细单(消费单编号,点餐单编号,菜品编号,单价，数量，下单时间)。

(4) 对于账单的局部 E-R 图，账单与餐台、操作员之间的联系，不需要单独建立表，把其他实体与账单实体的联系简化账单为外键，从而得到账单的关系模型如下：

账单(账单编号，点菜单编号，操作员编号，消费金额，结账时间)。

注：这里的双下划线表示主表中的外键。

4. 物理设计

数据库的物理结构设计是指根据数据库的逻辑结构来选定 RDBMS(如 Oracle、Sybase 等)，并设计和实施数据库的存储结构、存取方式等。

根据建立的关系模型，在 SQL SERVER2005 数据库管理系统中，创建 Restaurant 数据库，并参照表 6-2~表 6-8 的内容，使用表设计器依次创建各个表的结构。

表 6-2　TableInfo 餐台信息表

字段名	含义	类型	长度	**IS NULL**	主键	标识
TableID	餐台 ID	int		NOT NULL	是	是
TableName	餐台名称	nvarchar	50	NULL		

注：这是餐台固定信息表。餐台名称可以是：1 号台，2 号台…等。

表 6-3　FoodStyle 菜系表

字段名	含义	类型	长度	IS NULL	主键	标识
StyleID	菜系 ID	int		NOT NULL	是	是
StyleName	菜系名称	nvarchar	50	NULL		

表 6-4　FoodItem 菜品表

字段名	含义	类型	长度	**IS NULL**	主键	外键	标识
FoodID	菜品 ID	int		NOT NULL	是		是
SytleID	菜系 ID	int		NULL		是	
FoodName	菜品名称	nvarchar	50	NULL			
Price	单价	float		NULL			
ImagePath	图片位置	nvarchar	250	NULL			
IsHot	是否推荐	bit	1	NULL			
ClickCount	单击数	int					

注：FoodItem 表通过 StyleID 外键字段与 FoodStyle 表相关联。

表 6-5　Order　消费单

字段名	含义	类型	长度	**IS NULL**	主键	外键	标识
OrderID	消费单 ID	int		NOT NULL	是		是
TableID	餐台 ID	int		NULL		是	
Count	人数	int		NULL			
StartTime	开台时间	datetime		NULL			
Status	餐台状态	nvarchar	20	NULL			

注：Status 字段包含 2 种状态：开台、结账。

表 6-6　OrderDetail　消费明细单

字段名	含义	类型	长度	**IS NULL**	主键	外键	标识
OrderDetailID	明细单 ID	int		NOT NULL	是		是
OrderID	消费单 ID	int		NULL		是	
FoodID	菜品 ID	int		NULL		是	
Price	价格	float		NULL			
Count	数量	int		NULL			
OrderTime	下单时间	datetime		NULL			

注：OrderDetail 表通过 OrderID 字段与 Order 表产生关联。

表 6-7　Bill　结账单

字段名	含义	类型	长度	IS NULL	主键	外键	标识
BillID	账单 ID	int		NOT NULL	是		是
OrderID	消费单 ID	int		NULL		是	
UserID	操作员 ID	int		NULL		是	
Money	消费金额	float		NULL			
EndTime	结账时间	datetime		NULL			

表 6-8　Users　操作员表

字段名	含义	类型	长度	IS NULL	主键	外键	标识
UserID	操作员 ID	int		NOT NULL	是		是
UserName	操作员姓名	nvarchar	50	NULL			
Pwd	密码	int	10	NULL			
Phone	联系电话	float	50	NULL			

为了让读者对数据库 Restaurant 的各个表之间的关系有个整体的认识，给出各个表之间的关系示意图如图 6.11 所示。图 6.11 的表看上去比较多，其实可以按照这样的思路来理解：哪份点菜单(订单)，点了哪些菜（明细），消费金额是多少(结账)。实质上，这也是一个简单的订单管理数据库的内容。

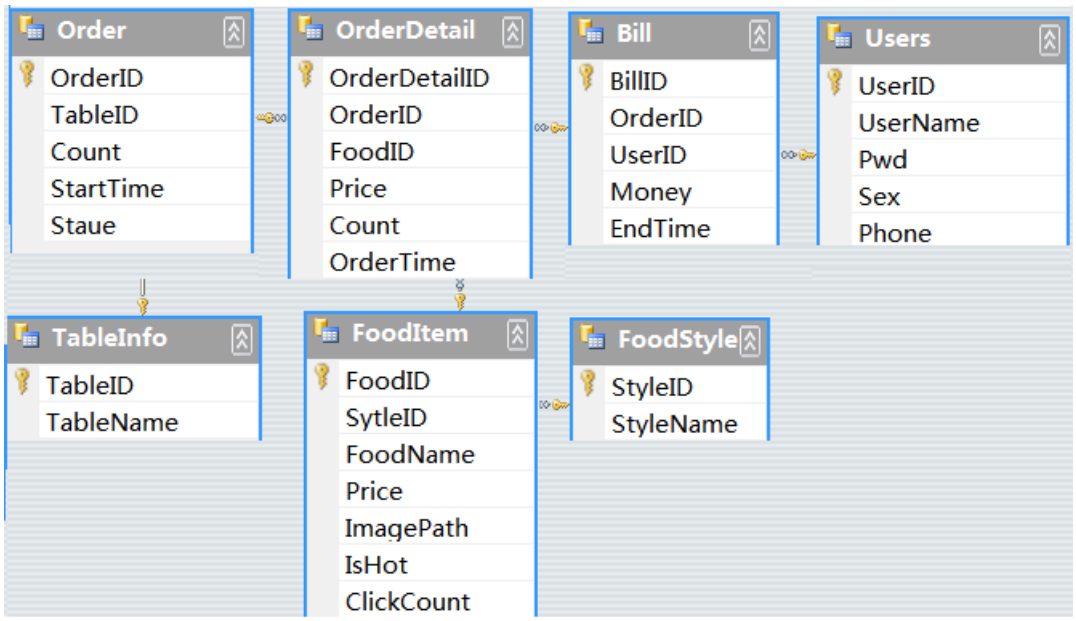

图 6.11　表之间关系示意图

参考-6.1.4 SQL 语句演练

本章所有的示例仅使用到 Restaurant 数据库中的菜品表 FoodItem 和菜系表 FoodStyle。在学习本节内容之前，请务必熟悉这两个表的结构(其他表可以在下一章学习时再回顾)，包括字段名、字段类型以及表之间的关系(具有完整性约束)。读者可以按照以上给出的表结构在 SQL Server Management Studio 中创建数据库，或附加本书附带的数据库样例(可以到清华

大学出版社网站下载本书的学习资源)。本节将简单介绍实现数据查询、插入数据、更新数据和删除数据的 SQL 语句的最基本的用法，这些语句也将在本章的实例中使用到。

1. 数据查询

基本语法

```
Select 子句 From 子句 [Where 子句]
```

应用实例 6.1 在菜品表 FoodItem 中，显示全部数据。

Select * From FoodItem

说明：为了测试查询结果，请启动 SQL Server Management Studio，在启动界面中选择“服务器类型”为“数据库引擎”，选择“服务器名称”为本机安装的数据库实例名，选择“身份验证”为“Windows 身份验证”方式登录，然后附加本书的 Restaurant 数据库。

在 SQL Server Management Studio 操作界面中单击工具栏中“新建查询”图标按钮，打开查询设计器界面，在查询设计器中输入上面的 Select 语句并执行，参照图 6.12 中标识出的步骤进行操作。

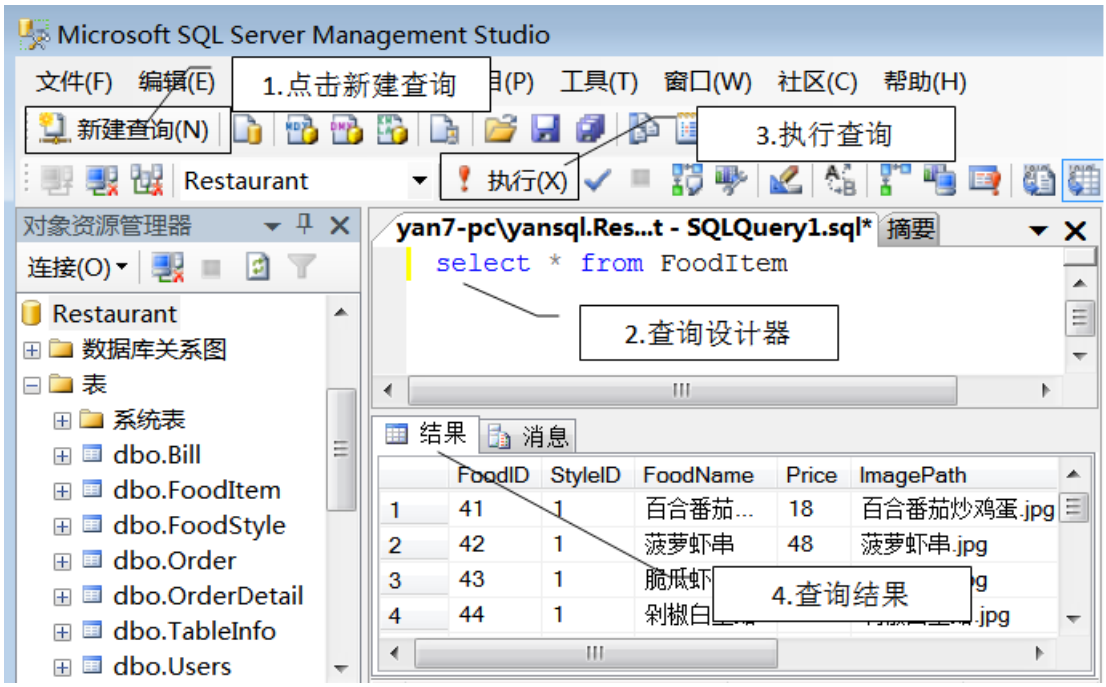

图 6.12　在查询设计器中测试 SQL 语句

应用实例 6.2 在菜品表 FoodItem 中，查询价格在 50~80 之间，StyleID 为 1 的菜品名和价格。

Select FoodName,Price From FoodItem Where (Price Between 50 And 80) And StyleID=1

说明：当按条件查询数据时，多个条件条件可以使用 And 和 Or 运算符。And 表示满足所有的条件，而 Or 表示满足任何一个条件；Between…And…表示数值范围，也可以使用>=和<=比较运算符来代替。多个条件时，每个条件最好使用括号作为独立的条件。

应用实例 6.3 在菜品表 FoodItem 中，显示菜品名包含“鸡”的菜品。

Select * From FoodItem Where FoodName Like '%鸡%'

说明：Like 表示模糊匹配，可以使用通配符“%”和“?”。“%”表示任意多个字符，而

“?”表示一个字符，字符串的值用一对单引号作为定界符。如果要精确匹配，请使用“=”运算符。

应用实例 6.4 在菜品表 FoodItem 中查询各种菜品类别的数量。

Select StyleID,Count(*) As 数量 From FoodItem Group By StyleID

说明：Group by 子句用于分组统计。用于分组统计的 Select 子句中，只能出现分组字段或统计结果字段。As 关键字是为查询结果字段赋予的别名。

应用实例 6.5 在菜品表 FoodItem 中，查询价格不为空值的菜品名和价格，并按价格升序，菜品名降序显示查询结果。

Select FoodName,Price From FoodItem WHERE Price Is Not NULL Order By Price Asc,FoodName Desc

说明：Order By 子句用于按指定的字段进行排序，Asc 表示升序，Desc 表示降序，默认为升序。多个字段排序时，每个字段用逗号分隔。Is Not NULL 用于查询不包含空值的字段，如果要查询包含空值的字段，则使用 Is NULL。

应用实例 6.6 查询菜品表 FoodItem 中所有菜品名及其所属的菜系名。

Select FoodItem.FoodName,FoodStyle.StyleName From FoodItem,FoodStyle Where FoodItem.StyleID=FoodStyle.StyleID

说明：要从多个表中选择数据，多个表之间必须存在某种关联。在这里，StyleID 字段既是 FoodItem 表中的外键，也是 FoodStyle 表的主键。可以根据键之间的联系得到两个表的交集。基本语法为如下：

```
Select 表 a.字段名,…,表 b.字段名，… From 表 a,表 b Where 表 a.X=表 b.X
```

其中，X 字段表示同名字段，或者是表之间存在关联的字段。多个表时，Where 子句用 And 运算符连接，如 Where 表 a.X 字段=表 b.X 字段 And 表 b.X 字段=表 c.X 字段。

也可以使用连接查询来实现，但上述语法比较简单易懂。使用连接查询的语句如下：

Select FoodItem.FoodName,FoodStyle.StyleName From FoodItem Inner Join FoodStyle *On* FoodItem.StyleID=FoodStyle.StyleID

2. 插入数据

基本语法

```
Insert Into 表名(字段列表) Values(值列表)
```

说明：字段列表是指表中一个或多个以逗号分隔的字段名；值列表是与字段列表对应位置和类型的具体数据。

应用实例 6.7 插入一条完整的记录到菜品表 FoodItem。

Insert Into FoodItem (StyleID,FoodName,Price,ImagePath,IsHot,ClickCount) Values(1,'美洲牛扒',12.6,'美洲牛扒.Jpg',1,1)

说明：FoodItem 表包含 int 类型的标识字段 FoodID，该字段是在添加记录时自动递增的，不应由用户添加；FoodItem 表还包含了约束字段 StyleID。注意，StyleID 是外键，也是 FoodStyle 表中的主键，在添加记录到 FoodItem 表时，由于表之间存在约束关系，StyleID 的值必须存在于 FoodStyle 表中(这里 StyleID 的值为 1)。对于字符串类型的字段，必须使用一对单引号作为定界符。FoodItem 表中的 IsHot 是 bit 类型字段，取值 0 或 1。

注意：Insert Into 语句中，字段列表和值列表在数量、位置和类型上必须一一对应；在 FoodItem 表插入数据时，StyleID 的值必须存在其作为主键的 FoodStyle 表中，这是由于表之间存在数据完整性约束。

3. 更新数据

基本语法

```
Update 表名 Set 字段名=字段值,字段名=字段值… [Where 子句]
```

说明：多个字段要同时更新时，每个字段的赋值表达式使用逗号分隔。如果带 Where 条件子句，那么将按条件更新表中的记录，否则，将更新表中所有记录。

应用实例 6.8 为菜品表 FoodItem 的所有菜品提价 10%。

```
Update FoodItem Set Price=Price+Price*0.1
```

说明：不含 Where 条件子句，所有的菜品都将在原来价格 Price 的基础上加 10%。

应用实例 6.9 修改菜品表 FoodItem 中 FoodID 为 10 的菜品，设置为推荐(IsHot 为 1)菜品，并使其单击数（ClickCount）加 1。

```
Update FoodItem Set IsHot=1,ClickCount=ClickCount+1 Where FoodID=10
```

说明：含 Where 子句，修改指定记录的字段。

4. 删除数据

基本语法

```
Delete From 表名 [Where 子句]
```

说明：如果不含 Where 子句，那么将删除整个表的所有记录，否则，仅删除满足条件的记录。

应用实例 6.10 删除菜品表 FoodItem 中价格低于 10,，而且是非推荐(IsHot 为 0)的菜品

```
Delete From FoodItem Where Price<10 and Ishot=0
```

说明：如果不含 Where 子句，那么将清空 FoodItem 表的所有数据，否则，删除满足指定条件的数据。

注意：如果要删除表的一条记录，而该表的主键是其他表的外键，那么必须删除其他表中所有包含该主键字段值的记录后方可执行删除，否则系统会拒绝删除，这也是数据完整性性约束的体现。如 FoodStyle 菜系表的主键 StyleID，它也是菜品表 FoodItem 的外键，如果要删除菜系表中 StyleID 值为 1 的记录，那么必须先删除菜品表中所有 StyleID 为 1 的记录，方可以删除菜系表的该条记录。

参考 6.2 使用 ADO.NET 数据库编程

6.2.1 什么是 ADO.NET

ADO.NET 是一组为.NET 程序员提供访问数据服务的类，其名称来源于.NET 技术出现之前的 ADO(ActiveX Data Object 数据访问对象)，但 ADO.NET 并非 ADO 的简单升级版，而是基于.NET 框架重新开发出来的新一代的数据处理技术，之所以沿用该名称，目的是为具有 ADO

开发经验的老程序员可以以类似使用 ADO 的经验快速熟悉和使用 ADO.NET。

ADO.NET 提供对关系数据、XML 和应用程序数据的访问，是.NET Framework 中不可缺少的一部分。ADO.NET 支持多种开发需求，包括创建由应用程序、工具、语言或 Internet 浏览器使用的前端数据库客户端和中间层业务对象。ADO.NET 类库位于 System.Data 命名空间下。System.Data 命名空间提供对 ADO.NET 结构的类的访问。通过 ADO.NET 可以生成一些组件，用于有效管理多个数据源的数据。

ADO.NET 主要包含两个重要的组件：.NET Framework 数据提供程序和数据集 DataSet。.NET Framework 数据提供程序用于连接到数据库、执行命令和检索结果，可以直接处理检索到的结果，或将其放入 DataSet 对象；DataSet 是一个或多个来自不同数据源的数据表及表关系的集合，存放在本地高速缓存，其数据可以通过.NET Framework 中的 DataAdapter 对象填充，或从 XML 文件导入，也可以到导出到 XML 文件。DataSet 对象也可以独立于.NET Framework 数据提供程序使用，以管理应用程序本地的数据或源自 XML 的数据。

图 6.13 参考了 Microsoft 公司给出的 ADO.NET 结构图，同时也说明了.NET Framework 数据提供程序与 DataSet 之间的关系。

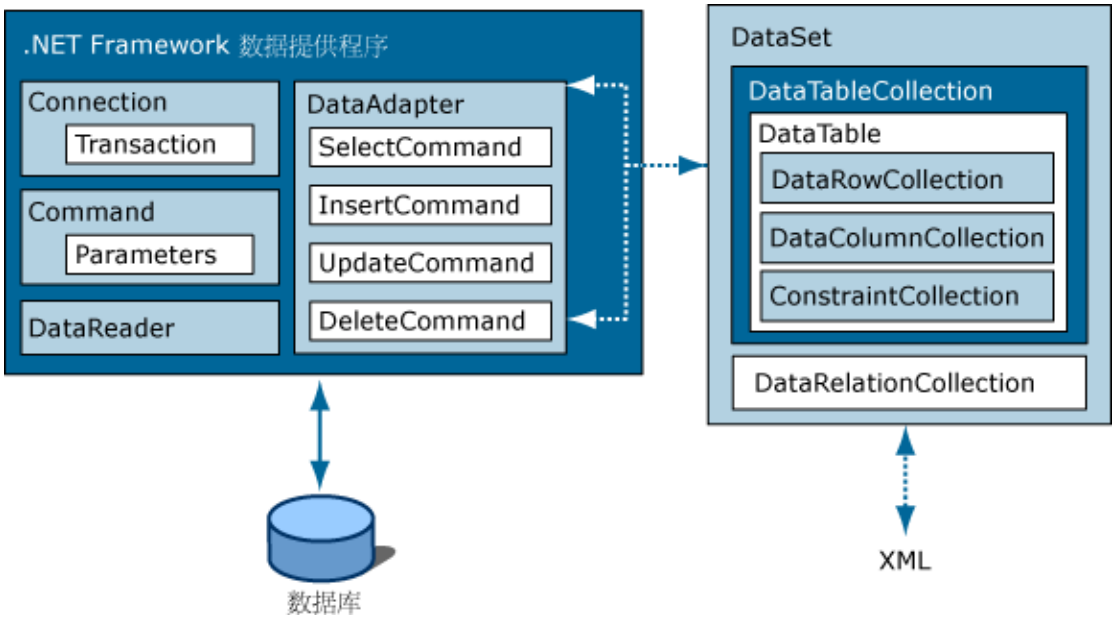

图 6.13　ADO.NET 结构

从图 6.13 可以看出，ADO.NET 主要包含两个组件，分别为.NET Framework 数据提供程序和数据集 DataSet。

对于目前主流的关系型数据库，.NET Framework 提供访问不同主流关系型数据库的.NET Framework 数据提供程序。表 6-9 列出了.NET Framework 中包含的.NET Framework 数据提供程序。

表 6-9 .NET Framework 数据提供程序

.NET Framework 数据提供程序	说明
SQL Server .NET Framework 数据提供程序	提供对 Microsoft SQL Server 7.0 版或更高版本的数据访问。使用 System.Data.SqlClient 命名空间
OLE DB .NET Framework 数据提供程序	适合于使用 OLE DB 公开的数据源。使用 System.Data.OleDb 命名空间

ODBC .NET Framework 数据提供程序	适合于使用 ODBC 公开的数据源。使用 System.Data.Odbc 命名空间
Oracle .NET Framework 数据提供程序	适用于 Oracle 数据源。Oracle .NET Framework 数据提供程序支持 Oracle 客户端软件 8.1.7 版和更高版本，使用 System.Data.OracleClient 命名空间

不同的.NET Framework 数据提供程序存在于不同的命名空间，用于访问不同的数据源，但核心功能和使用方法是一样的：连接数据源、执行命令、检索数据，或把数据读取到数据集进行处理。

6.2.2 SQL Server .NET Framework 数据提供程序

SQL Server .NET Framework 数据提供程序专门用于访问 SQL Server 数据库，位于 SqlClient 命名空间，其提供一组以 Sql 为前缀的类，如 SqlConnection、SqlCommand 和 SqlDataReader 等，具体内容见表 6-10。

表 6-10 SQL Server .NET Framework 数据提供程序主要的类

类	说　明
SqlConnection	连接到数据源
SqlCommand	表示要对数据源执行的 SQL 语句或存储过程
SqlParameter	表示 SqlCommand 的参数，还可以表示它到 DataSet 列的映射
SqlDataReader	提供从数据源读取数据行的只向前读写记录的方法
SqlDataAdapter	表示一组数据命令和一个数据库连接，用于填充 DataSet 和更新数据源
SqlException	在基础提供程序返回 OLE DB 数据源的警告或错误时引发的异常

从数据库读取数据的一般步骤如下：

(1) 创建并使用 SqlConnection 对象连接到数据库。

(2) 创建并使用 SqlCommand 对象执行 SQL 语句操作数据表。

(3) 创建并使用 SqlDataReader 对象读取数据；或创建并使用 SqlDataAdapter 对象操作数据，将数据填入 DataSet 对象。

(4) 显示数据。

注意：在决定应用程序使用 DataReader 或使用 DataSet 时，应考虑应用程序所需的功能类型。

DataSet 通常用于执行以下操作：

(1) 在应用程序中将数据缓存在本地，以便对数据进行处理。如果只需要读取查询结果，DataReader 是更好的选择。

(2) 在层间或从 XML Web 服务对数据进行远程处理。

(3) 与数据动态交互，例如，绑定到 Windows 窗体控件，或组合并关联来自多个源的数据。

(4) 对数据执行大量的处理，而不需要与数据源保持打开的连接，从而将该连接释放给其他客户端使用。

如果不需要 DataSet 提供的功能，可以用 DataReader 以只进、只读方式返回数据，从而

提高应用程序的性能。虽然 DataAdapter 使用 DataReader 填充 DataSet 的内容，但可以用 DataReader 提高性能。这样可以节省 DataSet 使用的内存，并将省去创建 DataSet 并填充其内容所需的处理。

下面详细介绍每一个对象的创建和使用。

6.2.3 创建和使用连接对象

要使用.NET Framework 数据提供程序访问 SQL Server 数据库，首先要在代码文件首部使用下面的语句声明命名空间的引用：

```
Using System.Data.SqlClient;
```

要实现对数据库的各种操作，必须首先使用连接对象连接上数据库。通过下面语句创建连接对象：

```
SqlConnection conn=new SqlConnection();
```

创建连接对象后，需要设置连接对象的连接字符串属性，以指明连接的数据源、登录验证方式和数据库名称。登录 SQL Server 有两种验证方式：① Windows 身份验证;② SQL server 身份验证。

(1) 如果采用 Windows 身份验证，可以设置连接字符串属性为：

```
Conn.ConnectionString=
”Data Source=(local);Initial Catalog=Restaurant; Integrated Security=true”;
```

其中，Data Source 设置数据库实例名称，如果使用本地数据库默认实例，可以简写为“(local)”或“.”(点号)；Initial Catalog 设置了初始数据库为 Restaurant；Integrated Security 设置为 true，表示使用 Windows 身份验证方式。

(2) 如果采用 SQL Server 身份验证，需要指明用户名 User id 和密码 Password，设置连接字符串属性为：

```
Conn.ConnectionString=
”Data Source=(loacal);Initial Catalog= Restaurant;User id=sa;Password=123”;
```

这里假定用户名为 sa，密码为 123。

创建连接对象并设置其连接字符串属性后，并未真正连接到数据库。要连接到数据库，需要调用其 Open 方法；当数据操作完毕后，需要调用其 Close 方法来关闭与数据库的连接，以释放占用的系统资源。

应用实例 6.11 使用 Windows 身份验证登录 SQL Server，与数据库 Restaurant 建立连接后随即关闭。

代码片段

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);Initial Catalog=Restaurant;
                         Integrated Security=true";
conn.Open();//打开连接
//在这里，执行其他数据库相关的数据表操作
```

```
conn.Close();//关闭数据库
```

代码说明

该代码片段演示了如何创建连接对象 conn，并通过连接对象的 Open 方法建立了与数据库的连接，未对数据库中的数据表进行任何相关的操作，随后使用连接对象的 Close 方法关闭与数据库连接。虽然没有实质的意义，但说明了连接数据库的过程，该过程也可以用来在操作数据库前测试是否可以连接上数据库，如果执行后不出现异常，则表示连接成功，否则表示连接失败。

如果使用该代码片段执行时出现异常，请检查：① SQL Server 数据库服务器是否启动。② 连接字符串的各个部分是否正确。

注意：Data Source 有多个别名，可以是 Server、Address、Addr、Network Adress 之一；Initial Catalog 也可以写为 Database；User ID 可以写为 UID；Password 可以写为 PWD，因此，下面语句与前面使用 SQL Server 身份验证的连接字符串是等价的：

```
Conn.ConnectionString=”Server=(local);Database= Restaurant;UID=sa;PWD=123”;
```

其中（local）也可以使用“.”来代替本地数据库服务器的实例名。如果没有密码可以留空，如：

```
Conn.ConnectionString=”Server=.;Database= Restaurant;UID=sa;PWD=”;
```

如果不使用数据库服务器的默认实例，则还需要指定实例名，如：

```
Conn.ConnectionString=”Server=.\\实例名;Database= Restaurant;UID=sa;PWD=”;
```

或：

```
Conn.ConnectionString=”Server=服务器名称(服务器 IP)\\实例名;Database=
Restaurant;UID=sa;PWD=”;
```

注意：Server 参数如果使用 IP 地址，需要在“SQL 配置管理器”应用程序中，为该数据库实例启用 TCP/IP 协议，并设定 IP 和访问端口。

为避免写错连接字符串各个部分的关键字，也为了方便记忆，可以使用连接字符串构建器 SqlConnectionStringBuilder 来帮助创建连接字符串，SqlConnectionStringBuilder 对象包含以下常用的属性：

(1) DataSource：用于指定数据源，等同于连接字符串中的Data Source的作用。

(2) InitialCatalog：指定登录后需要操作的数据库，等同于连接字符串中的Initial Catalog的作用。

(3) IntegratedSecurity：设置为true时表示使用Windows身份验证方式，设置为fasle时表示使用SQL Server身份验证方式。

(4) UserID ：使用SQL Server身份验证方式时的用户名。

(5) Password：使用SQL Server身份验证方式时的密码。

(6) ConnectionString：自动构建的连接字符串。

注意：无论采用哪种验证方式登录SQL Servre，均要设置DataSource和InitialCatalog属性。

应用实例 6.12 使用 SqlConnectionStringBuilder 对象构建连接字符串。

代码片段

```
SqlConnectionStringBuilder csb = new SqlConnectionStringBuilder();
csb.DataSource = ".";//1.指定数据源
```

```
csb.InitialCatalog = "Restaurant";//2.指定数据库
csb.IntegratedSecurity = true;//3.取为false时，需要下面被注释的两条语句
//csb.UserID = "sa";//4.用户名
//csb.Password = "123";//5.密码
SqlConnection conn = new SqlConnection();//6.创建连接对象
conn.ConnectionString = csb.ConnectionString;//7.设置连接字符串
```

代码说明

创建SqlConnectionStringBuilder对象后，首先设置其DataSource和InitialCatalog属性，指定数据源和初始数据库；然后根据验证方式的不同来设置其他属性，如设置IntegratedSecurity属性为true，代表使用Windows身份验证方式；或者设置UserID和Password属性，代表使用SQL Server身份验证方式；最后该对象通过前面设置好的属性自动构建ConnectionString属性，该属性代表建立好的连接字符串。这样就无须记住连接字符串的各个部分的关键字了。

总之，选择适合自己记忆的构建连接字符串的方式。

6.2.4 创建和使用执行命令对象

连接上数据库的目的之一是为了执行 SQL 命令以操作数据库中的数据。SqlCommand 命令对象用于执行 SQL 语句或存储过程。

1. 常用属性

(1) Connection：设置或取得已连接的对象。

(2) CommandText：要执行的 SQL 语句或存储过程。

(3) CommandType：用于表明 CommandText 是 SQL 语句还是存储过程，其值为 CommandType 类型枚举值之一，Text、TableDirect 或 StoredProcedure，默认值为 Text。

注意，只有用于 OLE DB .NET Framework 数据提供程序才支持 CommandType 取值为 TableDirect，表明 CommandText 直接取值为表名。

(4) Parameters：获取或设置 SQL 语句或存储过程中的命令参数的集合。

注意，SQL Serve .NET Framework 数据提供程序不支持在向通过 CommandType.Text 的命令调用的 SQL 语句或存储过程传递参数时使用“?”占位符。在这种情况下，必须使用命名参数。

2. 常用方法

(1) ExecuteScalar()：执行查询，并返回查询结果集中的第 1 行的第 1 列数据，忽略其他列或行。常用于统计行数、列计算或执行 SQL 语句后返回的结果，如插入新行时获取新行自动递增的标识字段。

(2) ExecuteReader()：将 CommandText 属性发送到 Connection 对象并返回一个数据读取器 SqlDataReader 对象。一般用于 Select 语句并可从返回的 SqlDataReader 对象中读取数据。

(3) ExecuteNonQuery()：执行 SQL 语句并返回受影响的行数，一般用于执行包含 Update、Insert、Delete 的 SQL 语句。

3. 创建命令对象方式

创建 SqlCommand 对象的常用方法如下几种：

(1) 使用带参数的构造方法，同时指定 SQL 命令文本和连接对象。

```
string sql=”Select * from 表名”;
SqlCommand comm = new SqlCommand(sql,conn）
```

本书采用这种方式。这里 conn 代表已创建的连接对象。

(2) 通过已经创建的连接对象 conn 的 CreateCommand 方法得到，如：

```
SqlCommand comm=conn. CreateCommand ();
```

这种方式创建 SqlCommand 对象后，需要为其设置 CommandText 属性，以执行相关操作。

(3) 使用无参数构造方法创建。

```
SqlCommand comm = new SqlCommand();
```

这种方式创建 SqlCommand 对象后，需要为其设置 Connection 和 CommandText 属性，以执行相关操作。

4. 使用命令对象

创建 SqlCommand 对象并设置好相关属性后，可以执行对数据库的各种操作。

注意，下面所有的实例都需要引入System.Data.SqlClient命名空间；而且需要在SQL Server Management Studio中附加本书所带的Restarant数据库（或参照6.1.3节自行创建该数据库）。另外，读者在上机测试各个应用实例代码时(应用实例6.13~6.16)，请自行创建C#项目，在默认窗体中添加一个Button按钮，并在该按钮的事件处理过程中输入相关代码。

应用实例 6.13 取得菜系表 FoodStyle 的记录数。

代码片段

```
//1.构建连接字符串
string connString = "Data Source=.;Database= Restarant;Integrated Security=true";
SqlConnection conn = new SqlConnection(connString);//2.创建连接对象
conn.Open();//3.打开连接

string sql = "select count(*) from FoodStyle";//4.Select查询的命令文本
SqlCommand comm = new SqlCommand(sql, conn);//5.创建命令对象
int count=(int)comm.ExecuteScalar();//6.执行查询
conn.Close();//7.关闭连接
```

代码说明

(1) 注释 1：使用 Windows 身份验证方式构建连接字符串，并保存到字符串变量 connString 中。

(2) 注释 2、注释 3：使用连接字符串变量 connString 作为参数来创建连接对象 conn，然后调用其 Open 方法实现打开与数据库的连接。

(3) 注释 4：构建用于统计 FoodStyle 表中记录数的 SQL 命令文本。

(4) 注释 5：使用命令文本 sql 和连接对象 conn 作为参数，创建命令对象 comm。

(5) 注释 6：ExecuteScalar 方法用于取得单值查询结果(该方法只有返回一个值)，由于其返回值为 object 类型，因此需要强制转换为 int 类型。返回结果的具体类型，是根据 SQL 语句中指定的查询结果而定的。

ExecuteScalar 方法也可以在插入数据时，得到标识列的值。

应用实例 6.14 向菜系表 FoodStyle 插入一条记录，并取得该记录标识列的值。

代码片段

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);Initial Catalog=restaurant;Integrated
  security=true";
conn.Open();//打开连接
//执行数据库操作
string sql = "Insert Into FoodStyle(StyleName) Output Inserted.StyleID Values('汤类')";
SqlCommand comm = new SqlCommand(sql, conn);
int StyleID=(int)comm.ExecuteScalar();
conn.Close();//关闭数据库
MessageBox.Show("新记录的ID是："+StyleID);
```

代码说明

本例与应用实例6.13不同的地方在于查询字符串sql的内容。其中，查询字符串中的values关键字前的语句Output Inserted.StyleID表示在执行插入数据操作后，返回新插入记录的标识列(自动编号列)StyleID的值。这里Output Inserted是SQL语句中的关键字，必须位于Values关键字之前。注意Inserted关键字后有连接符“.”，后紧跟标识列列名。

应用实例 6.15 在 Restaurant 数据库的菜品表 FoodItem 中插入一条记录。

代码片段

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);Initial Catalog=restaurant;Integrated
security=true";
conn.Open();//打开连接
//执行数据库操作
string sql = "Insert Into FoodItem(StyleID,FoodName,Price,ImagePath,IsHot) ";
sql += " Values(1,'番茄炒蛋',68,'番茄炒蛋.jpg',1)";

SqlCommand comm = new SqlCommand(sql, conn);
comm.ExecuteNonQuery();
conn.Close();//关闭数据库
```

[代码说明]

(1) 本例是执行非 Select 查询的 SQL 语句，实现向 FoodItem 表中添加 1 条记录。与应用实例 6.14 不同的地方是调用了 ExecuteNonQuery 方法来实现，但执行结果与调用 ExecuteScalar 方法没有区别。注意 SQL 命令文本中的字符串类型数据需要一对单引号作为定界符。

(2) 通常使用 ExecuteNonQuery 方法来执行 SQL 的 Insert、Update 和 Delete 语句，该方法返回一个 int 类型的值，代表执行这些操作后受影响的记录数；虽然使用 ExecuteScalar 方法也可以达到相同的效果，但更多的时候 ExecuteScalar 方法用于需要获取单值查询结果的 Select 语句。

(3) 为减少拼接字符串产生错误，上面粗体部分的代码，也可以使用命令参数方式来代替，如下面的代码：

```
//1.带命名参数的SQL命令文本
string sql = "Insert Into FoodItem(StyleID,FoodName,Price,ImagePath,IsHot) ";
sql += " Values(@StyleID,@FoodName,@Price,@ImagePath,@IsHot)";
//2.创建命令对象
SqlCommand comm = new SqlCommand(sql, conn);
//3.为命名参数指定具体值
comm.Parameters.AddWithValue("@styleID", 1);
comm.Parameters.AddWithValue("@FoodName", "番茄炒蛋");
comm.Parameters.AddWithValue("@Price", 68);
comm.Parameters.AddWithValue("@ImagePath", "番茄炒蛋.JPG");
comm.Parameters.AddWithValue("@IsHot", 1);
//3.执行查询
comm.ExecuteNonQuery();
```

代码说明

(1) 注释1：构建带命名参数的SQL命令文本，@后面的名称表示SQL命令文本中的命名参数名，命名参数名可以是符合变量命名规则的任意名称，这里的命名参数与字段同名，主要是为了使用直观、方便，避免漏写参数。

(2) 注释 2：Parameters 是命令对象的集合属性，代表命名参数集。AddWithValue 方法用于为指定的命名参数提供具体值，方法中的参数 1 是字符串类型的命名参数，注意参数名前带@，与 SQL 命令文本中的命名参数对应；参数 2 是为命名参数提供类型一致的值。使用 AddWithValue 方法为 SQL 命令文本中的命名参数赋值，可以降低因需要构建 SQL 命令字符串带来的复杂度。

Parameters 集合属性的每个元素都是 Parameter 类型的对象，可以单独创建 Parameter 对象并添加到该集合中，但使用其 AddWithValue 的方法更为简单。

不带命名参数的 SQL 命令文本写法上比较简洁，但容易拼接出错；带命名参数的 SQL 命令文本使用简单，直观，不容易出错，但代码量大。读者可以选择适合自己的使用方式。

应用实例 6.16 在菜品表 FoodItem 中，将菜名为“番茄炒蛋”的价格(Price)更新为 90。

代码片段

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);Initial Catalog=restaurant;Integrated
  security=true";
conn.Open();//打开连接
string sql = "Update FoodItem Set Price=@Price Where FoodName=@FoodName";
SqlCommand comm = new SqlCommand(sql, conn);
comm.Parameters.AddWithValue("@Price", 98);
comm.Parameters.AddWithValue("@FoodName", "番茄炒蛋");
comm.ExecuteNonQuery();
conn.Close();//关闭数据库
```

代码说明：与应用实例 6.15 实现的方式类似，使用了带参数的命令文本，并在创建命令对象后，使用其集合属性 Parameters 的 AddWithValue 方法为各个命名参数赋值。

6.2.5 使用 SqlDataReader 读取数据

数据读取器SqlDataReader用于快速、向前读取查询结果集中的数据。该对象无法使用new方式创建，但可以从命令对象的ExecuteReader方法的返回值中得到。

1. 常用属性

(1) FieldCount：获取当前行中的列数。

(2) HasRows：获取一个值，该值表示 SqlDataReader 是否包含一行或多行。

(3) Item[列下标/列名]：SqlDataReader 对象的索引器。

2. 常用方法

(1) GetName(int index)：获取指定列的名称。

(2) GetValue(int index)：获取指定列的值。

(3) IsDBNull(int index)：判断指定列是否包含不存在或缺少的值。

(4) Read()：使 SqlDataReader 对象前进到下一条记录。

应用实例 6.17 获取菜品表 FoodItem 的表结构(所有的列名)，以及前 5 条记录。

新建项目，保存为 MyDataReader。

界面布局

在 Form1 窗体中添加 1 个标签和 1 个按钮，默认控件的名称，界面布局如图 6.14 所示。

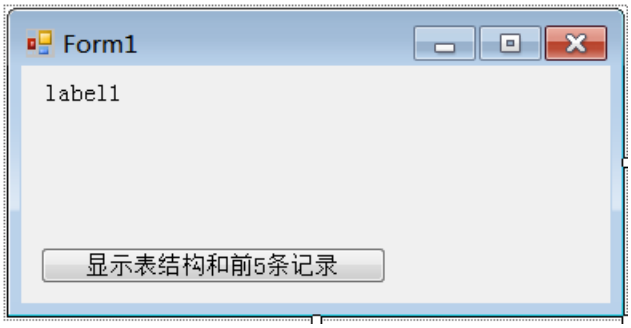

图 6.14　界面布局

程序代码

```
//省略其他命名空间声明部分
using System.Data.SqlClient;//必须该引用命名空间

namespace MyDataReader
{
    public partial class Form1 : Form
    {
        public Form1() {     InitializeComponent();     }//构造方法
        //按钮button1的事件处理过程
        private void button1_Click(object sender, EventArgs e)
        {
            //1.创建连接对象，并打开连接
```

```
            SqlConnection conn = new SqlConnection();
            conn.ConnectionString = "Data Source=(local);Initial Catalog= restaurant;
            Integrated security=true";
            conn.Open();

            //2.创建命令对象
            string sql = "select top 5 * from FoodItem";
            SqlCommand comm = new SqlCommand(sql, conn);

            //3.执行命令，得到SqlDataReader对象
            SqlDataReader dr = comm.ExecuteReader();

            string fields = "", values = "";//定义保存字段名和行数据的变量
            //4.取得表结构
            for (int i = 0; i < dr.FieldCount; i++)
            {
                fields += dr.GetName(i) + " ";
            }

            //5.取得每一行所有字段的数据，连接成字符串
            while (dr.Read())
            {
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    values += dr.GetValue(i) + " ";
                }
                values += "\r\n";
            }

            dr.Close();//6.关闭数据读取器对象
            conn.Close();//7.关闭连接对象
            label1.Text = fields + "\r\n" + values;
        }
    }//Form1类结束
}//命名空间结束
```

代码说明

(1) 本例实现在单击按钮 button1 时，取得表 FoodItem 的表结构和前 5 条记录并显示到标签 label1 中。

(2) 注释2：SQL命令文本中的top 5表示取得表中前5条记录，“*”表示所有字段。如果要取得FoodItem表中前20%的记录，可以写为top 20 percent，percent关键字表示百分比，如：

```
Select top 20 Percent * From FoodItem
```

(3) 注释 3：通过命令对象的 ExecuteReader 方法，执行 Sql 查询命令，返回 SqlDataReader 对象。ExecuteReader 方法仅用于执行 Select 查询，并返回 SqlDataReader 数据读取器对象。

(4) 注释 4：FieldCount 属性表示取得表中字段数(列数)，它是根据 Select 语句中指定的字段数来确定的。GetName 方法表示取得下标为 i 的列名(字段名)。

(5) 注释 5：Read 方法表示将记录指针移动到下一行，准备读取该行数据。注意：刚取得的 dr 对象不指向任何行，第一次调用时才指向第 1 行，所以至少要调用一次 Read 方法才能开始读取记录中字段值。如果返回值为 true，表示存在行数据，否则，表示数据读取完毕或没有数据；GetValue 方法用于取得指定列的值(字段值)，也可以使用 dr[i]或 dr[列名]的方式来代替，结果也一样。

(6) 注释 6、注释 7：数据读取完毕后需要关闭 dr 和 conn 对象，以断开与数据库的连接。

最常用的方法是通过列名来取得当前行中的字段值，如 dr[列名]。上面代码中的 while 循环块可以使用下面代码来代替：

```
while (dr.Read())
{
    values += dr["FoodId"]+" "+ dr["StyleID"]+" "+ dr["FoodName"]+" "+
    values += dr["Price"]+" "+ dr["ImagePath"]+" "+ dr["IsHot"]+" "+ dr["ClickCount"];
    values += "\r\n";
}
```

这样，即使以后修改了表中列的顺序，代码也无需改变。本章后面的例子均采用这种方法来获取字段值；而通过 dr[下标]或 GetValue 方法方式取得字段值，其顺序与 Select 查询中的字段列表顺序或表结构中字段顺序有关。

运行结果

按 F5 键运行程序，并单击界面上的按钮，运行结果类似图 6.15 所示。

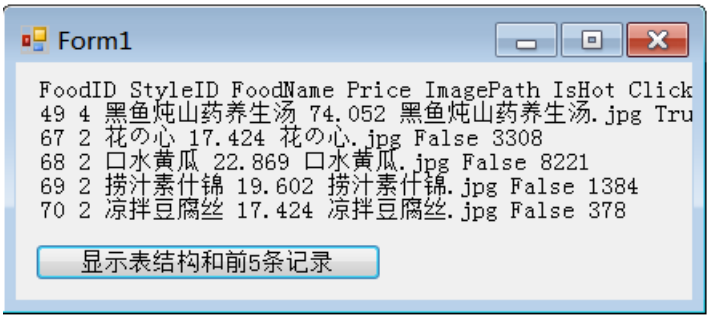

图 6.15　运行结果

典型案例：点餐系统前台设计

在网络点餐管理系统中，顾客可以通过餐台配备的触摸屏进行点餐。本例将设计一个用于顾客点餐的前台界面，以图文的形式显示从菜品表 FoodItem 读取出来的信息，这些信息包括菜品名称、图片和价格。顾客点餐界面的运行效果如图 6.16 所示。

1. 案例分析

本书在第 4 章“应用实例 4.3 以图文形式，设计一个顾客点餐界面”的例子中，曾介绍流式布局面板控件 FlowLayoutPanel 的使用以及如何将标签作为子控件动态添加到该控件的具体实现，本例也将以同样的方式实现点餐界面，只是标签的内容不再来自给定的字符串数组，而是动态从数据库中读取出来。

要从数据库中读取数据，首先要创建连接对象连接到数据库，然后使用命令对象执行 Select 查询命令后得到数据读写器对象，通过数据读写器对象的相关方法实现对数据的读取，在“应用实例 6.17”中已经介绍了完整的读取数据的过程。

图 6.16　运行效果

2. 实现步骤

新建项目，保存为 MyFoodMenu。

界面布局

在 Form1 窗体中放置 1 个 Panel 控件作为操作面板，设置其 Dock 属性为 Bottom，让其停靠在窗体底部；添加 2 个按钮作为 Panel 控件的子控件；再添加 1 个 FlowLayoutPanel 控件到窗体，设置其 Dock 属性 Fill，填满窗体剩余部分，设置其 AutoScroll 属性为 True，实现自动出现滚动条。保留各个控件默认的名称，界面布局如图 6.17 所示。

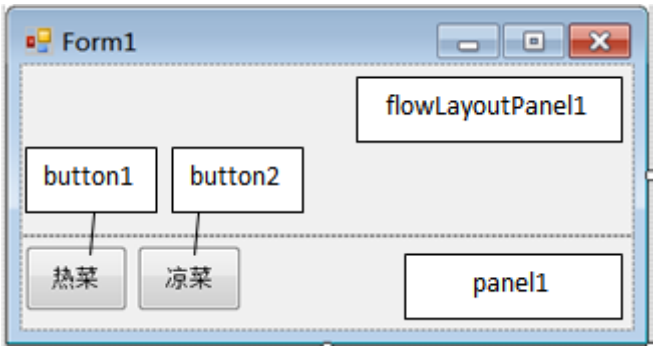

图 6.17　界面布局

在项目中添加一个 FoodItem 类文件，该类主要用于暂存从 FoodItem 表中读取出来的行数据，目的是在窗体之间或者方法之间将行数据作为参数进行传递，否则，要实现参数传递就需要定义多个变量来保存行数据。

程序代码 - FoodItem.cs 文件

```
//省略 using 命名空间默认添加的部分
```

```
namespace MyFoodMenu
{
    public class FoodItem
    {
        public int FoodID { get; set; }//菜品ID
        public string FoodName { get; set; }//菜品名称
        public float Price { get; set; }//价格
        public string ImagePath { get; set; }//图片位置
    }
}
```

代码说明：在FoodItem类中添加与菜品表FoodItem列名相同的属性，目的使用起来方便。属性的个数根据需要保存的数据而定，这里仅仅需要保存菜品 ID、菜品名称、价格和图片位置，也可以定义与表字段数完全一致的更多属性，这样的类也叫实体类。

程序代码 - Form1.cs 文件

```
//省略其他 using 命名空间默认添加的部分
using System.Data.SqlClient;
using System.IO;

namespace MyFoodMenu
{
    public partial class Form1 : Form
    {
        public Form1()  {  InitializeComponent();  }//构造方法
        //自定义方法1：查询FoodItem表的信息
        void GetFood(int styleID)
        {
            //1.创建连接对象，并打开与数据库的连接
            string connString = "Server=(local);DataBase=restaurant;
            Integrated Security =true";
            SqlConnection conn = new SqlConnection(connString);
            conn.Open();

            //2.根据条件，执行Select查询，得到查询结果集
            string sql = "Select * From FoodItem Where StyleID=" + styleID;
            SqlCommand comm = new SqlCommand(sql, conn);
            SqlDataReader dr = comm.ExecuteReader();

            //3.创建对象，准备保存读取的数据
            FoodItem Food = new FoodItem();

            //4.依次读取每一行记录，并将行数据作为参数，传递到ShowFood方法去显示
            while (dr.Read())
            {
```

```
            //读取菜品ID、名称、价格、图片位置
            Food.FoodID = int.Parse(dr["FoodID"].ToString());
            Food.FoodName = dr["FoodName"].ToString();
            Food.Price = float.Parse(dr["Price"].ToString());
            Food.ImagePath = dr["ImagePath"].ToString();

            ShowFood(Food);//将信息添加到界面
        }

        //5.关闭与数据库的一切连接
        dr.Close();
        conn.Close();
}

    //自定义方法2：显示数据到界面
    void ShowFood(FoodItem Food)
    {
        //6.动态创建Label对象，设置必要的属性
        Label lb = new Label();
        lb.Name = "ID" + Food.FoodID;
        lb.Text = Food.FoodName + "\r\n" + Food.Price;
        lb.Click += new EventHandler(lb_Click);//7.关联事件处理过程

        //8.动态设置Label的外观项
        lb.AutoSize = false;
        lb.Size = new Size(140, 120);
        lb.BackColor = Color.Black;
        lb.ForeColor = Color.White;
        lb.Margin = new System.Windows.Forms.Padding(6);
        lb.ImageAlign = ContentAlignment.TopCenter;
        lb.TextAlign = ContentAlignment.BottomCenter;

        //9.判断图片文件是否存在，如果存在，设置为Label的图像
        if (File.Exists(Application.StartupPath + "\\Images\\" +
        Food.ImagePath))
        {
            Image Img= Image.FromFile(Application.StartupPath +
        "\\Images\\" + Food.ImagePath);
            Bitmap bm = new Bitmap(img,130, 80);//目的：转换为合适尺寸
            lb.Image = bm;
        }

        //10.将Label添加到流式布局面板控件
        flowLayoutPanel1.Controls.Add(lb);
```

```
        }

        //11.动态标签的Click事件处理过程
        void lb_Click(object sender, EventArgs e)
        {
            Label lb = sender as Label;
            MessageBox.Show(lb.Name+"\r\n"+lb.Text);
        }

        //12.Load事件处理过程：默认显示热菜(StyleID=1)数据
        private void Form1_Load(object sender, EventArgs e)
        {
            GetFood(1);
        }

        //13."热菜"按钮button1的事件处理过程
        private void button1_Click(object sender, EventArgs e)
        {
            flowLayoutPanel1.Controls.Clear();//为更新控件显示，清空子控件
            GetFood(1);
        }

        //14."凉菜"按钮button2的事件处理过程
        private void button2_Click(object sender, EventArgs e)
        {
            flowLayoutPanel1.Controls.Clear();
            GetFood(2);
        }
    }//Form1类结束
}//命名空间结束
```

代码说明

(1) 在Form1窗体类文件中,设计了两个自定义方法GetFood和ShowFood。方法GetFood实现根据代表菜系编号的参数StyleID，对菜品表FoodItem的数据进行筛选，每读取一条记录，保存到自定义类FoodItem的对象Food中，并将Food对象作为自定义方法ShowFood的参数；在ShowFood方法中，为将菜品信息显示到窗体界面中，为每一个菜品动态创建一个Label对象lb，设置其文本内容和其他外观属性后作为子控件添加到flowLayoutPanel1控件中。

(2) 注释1：使用Windows身份验证方式，创建连接对象conn，并打开与Restaurant数据库的连接。

(3) 注释2：根据查询条件构建命令字符串sql，通过sql和连接对象conn作为参数创建命令对象comm，将comm对象调用ExecuteReader方法执行查询后的返回值保存到数据读取器对象dr中。

(4) 注释 3：创建 FoodItem 类型对象 Food，准备使用其属性保存从表中读取的数据。

(5) 注释 4：在循环中，通过 dr 对象的 Read 方法读取查询结果中的每一行记录，并将当前读取的记录保存到 Food 对象对应的属性中。由于使用 dr["字段名"]方式得到的字段值是对象类型，因此需要转换为属性对应的类型进行保存。每读取一行记录，都将保存了记录信息的 Food 对象作为参数，调用 ShowFood 方法。

(6) 注释 5：关闭与数据库的连接，必须先关闭 dr 对象，再关闭 conn 对象。

(7) 注释 6：创建标签 Label 对象，设置其 Name 属性，Name 属性是区分对象的唯一标识，只要不重复即可。在 Text 属性中将菜品名和价格分行显示。

(8) 注释 7：为动态创建的标签对象关联同一事件处理过程，在该过程中可以通过标签对象的 Name 属性检测用户单击了哪个对象。Click 事件处理过程的结构可以自动创建，方法是在输入+=之后，按两次 Tab 键。

(9) 注释 8：设置 Label 对象的外观属性，使其尺寸可以调整，具有黑底白字并居中对齐的效果。

(10) 注释9：首先判断了图片文件是否存在，然后装载图片文件得到Image对象。

① 装载图片文件使用的语句如下：

```
Image.FromFile(Application.StartupPath + "\\Images\\" + Food.ImagePath );
```

该语句实现从应用程序当前运行位置的Images文件夹中装载指定的图片文件，返回Image对象。图片文件名保存在Food.ImagePath属性中，Application.StartupPath表示应用程序当前的运行位置，该位置必须包含Images子文件夹，且Images文件夹必须存在与菜品对应的图片文件，否则执行该语句时将出现“文件找不到”的异常。

② 创建Bitmap对象使用的语句如下：

```
Bitmap bm = new Bitmap(img,130, 80);
```

这是创建Bitmap位图类对象时使用的构造方法之一，参数1为Image对象，参数2为宽度，参数3为高度。通过给定的宽度和高度，对Image对象进行缩放，得到新的位图对象bm。Bitmap类继承于Image类，因此位图对象bm也可以用于设置控件的Image属性，从而显示缩放后的图片。如果不进行图像缩放，Label控件将显示原始图片的大小。

(11) 注释11：标签的Click事件处理过程。这里所有动态创建的标签都使用同一事件处理过程，参数sender代表当前触发事件的对象，使用as强制转换为Label类型，目的是可以取得标签的属性，如Name和Text属性，可以用于判断当前触发事件的是哪个标签。

(12) 注释12~注释14：为了刷新当前界面显示，需要先执行清空FlowLayoutPanel控件的操作，然后再调用GetFood方法，重新执行查询并显示数据。

参考-6.2.6 数据表格控件 DataGridView

数据表格视图控件 DataGridView 用于以表格的方式直观显示数据，也可以直接在表格中对数据进行编辑操作。而在实际开发中，通常使用表格以只读的方式显示数据，而数据的编辑操作将在新窗体中进行，这样做的目的是直观显示当前要编辑的数据，方便在更新数据时对数据进行有效性验证，通过验证的数据将同时更新表格和数据库。DataGridView 控件通常称为表格控件或表格，该控件位于工具箱的“数据”组。本节主要介绍如何创建表格结构、添加新行以及如何填充行数据。

1. 常用属性

DataGridView 控件常用的属性如表 6-11 所示。

表 6-11 DataGridView 控件常用属性

名称	说明
AllowUserToAddRows	读/写，该值指示是否向用户显示添加行的选项
AllowUserToDeleteRows	读/写，该值指示是否允许用户删除行
AllowUserToOrderColumns	读/写，该值指示是否允许通过手动对列重新定位
AllowUserToResizeColumns	读/写，该值指示用户是否可以调整列的大小
AllowUserToResizeRows	读/写，该值指示用户是否可以调整行的大小
Columns	获取一个包含控件中所有列的集合
CurrentCell	获取或设置当前处于活动状态的单元格
CurrentRow	获取包含当前单元格的行
MultiSelect	读/写，该值指示是否允许用户一次选择的多个单元格
ReadOnly	读/写，该值指示用户是否可以编辑单元格
RowHeadersVisible	读/写，该值指示是否显示包含行标题的列
Rows	获取一个集合，该集合包含所有行
SelectionMode	读/写，该值指示如何选择单元格

2. 常用方法

DataGridView 控件的常用方法主要包含在其行集合 Rows 和列集合 Columns 属性中，通过这两个集合属性的相关方法来实现对表格的行和列操作。

(1) 清空表格列：Columns.Clear()。

(2) 添加列：Columns.Add(列名，列标题)。

(3) 移除指定的表格列：Columns.RemoveAt(列下标)或Columns. Remove(列名)。

(4) 清空表格行：Rows.Clear()。

(5) 移除指定的表格行：Rows.RemoveAt(行下标)或Rows.Remove(行对象)。

(6) 添加空行：Rows.Add()。

3. 常用事件

DataGridView 控件常用的事件为单击或双击单元格时触发的事件。

(1) CellClick：单击单元格时触发的事件。

(2) CellDoubleClick：双击单元格时触发的事件。

DataGridView 控件拖放到窗体时默认是没有行和列的空表格，要显示数据，首先要创建表格列，即表结构。可以通过代码来动态创建表格列，但通常做法是在其属性窗口的 Columns 属性中，通过向导来创建表格列。假如在窗体中添加了一个 DataGridView 控件，通过向导创建表格列的步骤如下：

(1) 选中 DataGridView 控件，单击在其右上角出现的智能标记，选择“添加列”；或者在属性窗体中找到 Columns 属性，单击其右边的“…”按钮，如图 6.18 所示，都将打开如图 6.19 的“添加列”对话框。

(2) 在“添加列”对话框中的“名称”栏输入列名，一般与表的字段同名，该名称即为列的 Name 属性，也是引用表格单元格的依据；在“页眉文本”栏输入该列的标题文字，目的是让用户理解该列所代表的含义，该标题即为列的 HeaderText 属性，相当控件的 Text

属性。依次添加所需的列，直到完成所有列的添加。

(3) 如果要修改列属性，或者对列进行更多的属性设置，比如宽度、高度、背景色或前景色等，可以使用在图 6.18 中标示的方法 1，选择“编辑列“；或使用在图 6.18 中标示的方法 2，都将打开“列编辑”窗体，在该窗体完成所需的操作。

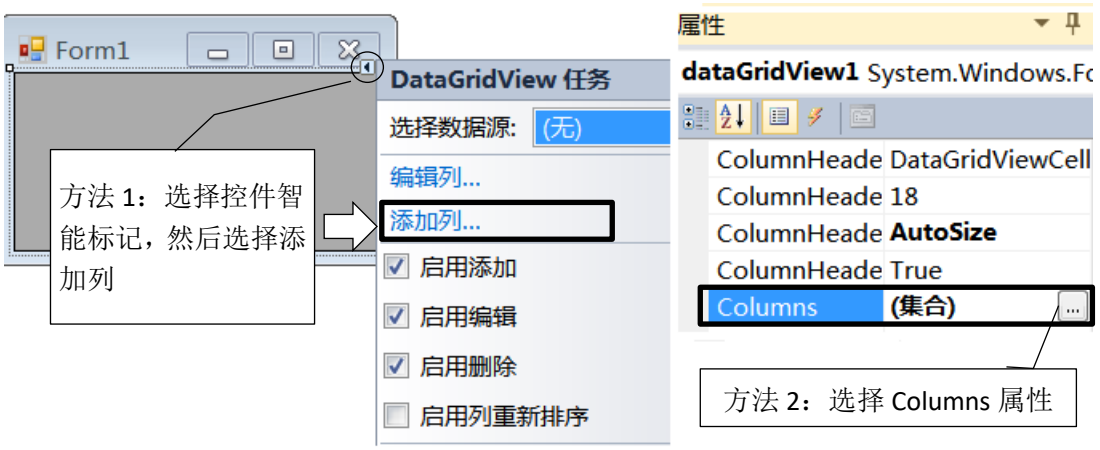

图 6.18　打开“添加列”对话框的两种方法

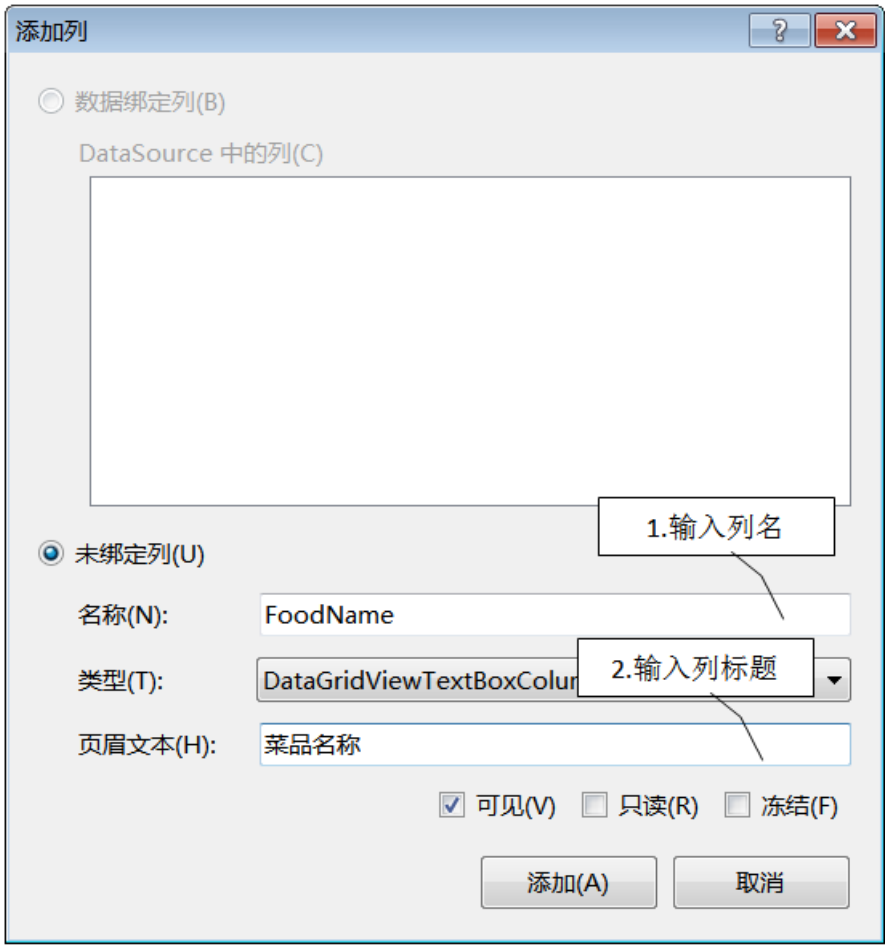

图 6.19　添加列对话框

如果要通过代码来动态创建表格列，并设置列的宽度，可以使用以下的代码片段，该代码片段用于创建包含 7 列的表格，并设置合适的列宽度。这里假定 DataridView 控件的名称

为 dgv。

代码片段

```
//1.创建表格列
dgv.Columns.Add("Pos", "序号");//参数分别为列名，列标题
dgv.Columns.Add("StyleID", "菜系ID");
dgv.Columns.Add("StyleName", "菜系名称");
dgv.Columns.Add("FoodID", "菜品ID");
dgv.Columns.Add("FoodName", "菜品名称");
dgv.Columns.Add("Price", "价格");
dgv.Columns.Add("ImagePath", "图片位置");

//2.设置列宽
dgv.Columns[0].Width = 80;
dgv.Columns[1].Width = 80;
dgv.Columns[2].Width = 160;
dgv.Columns[3].Width = 80;
dgv.Columns[4].Width = 160;
dgv.Columns[5].Width = 100;
dgv.Columns[6].Width = 160;
```

代码说明

(1) 表格集合属性 Columns 的 Add 方法实现为表格添加了一个列名为 Pos、列标题为“序号”的列，方法中的参数 1 表示列名，参数 2 表示列标题。

(2) 使用表格属性Columns的列下标或列名的方式引用列，如Columns[0]代表第1列，下标从0开始，列的下标与创建列的顺序有关；同样，Columns["Pos"]也代表第1列。

无论是通过向导方式或通过代码方式创建了表格列，数据的填充都必须通过代码使用 Rows 集合的方法来实现。假如窗体添加了控件 dgv，并创建了表格列，下面的代码实现向表格 dgv 添加数据行并填充单元格数据：

```
int count = dgv.Rows.Count;//1.取得当前行数
dgv.Rows.Add(); //2.添加空行
dgv.Rows[count].Cells["Pos"].Value = count + 1;//3.填充单元格数据
dgv.Rows[count].Cells["FoodName"].Value ="番茄炒蛋";//4. 填充单元格数据
```

上面给出的代码中，注释1前的语句表示取得当前表格的行数；注释2前的语句表示在表格末尾添加一个空行；注释3前的语句表示向表格最后一行(新创建的空行)列名为Pos的单元格填充数据。Cells是集合属性，代表指定行的单元格的集合，可以通过列下标或列名引用单元格，如Cells[0]代表第1个单元格。一般使用列名引用单元格，因为列名与列的顺序无关，如Cells["Pos"]代表列名为Pos的单元格。单元格的Value属性代表单元格的值，可以读写。注释3和注释4都是向新行的单元格赋值，从而将数据显示到单元格。

由于表格控件在创建好表格结构后，底部默认带有一个空行，目的是允许用户直接添加数据，此时表格属性AllowUserToAddRows的默认值为true。在使用代码添加新行，或者用户在直接原有的空行输入数据的同时表格又会自动产生新的空行。因此，执行上面的代码前，需要先将dgv控件属性AllowUserToAddRows属性设置false，即不允许用户在表格中直接添加新行数据，而是由代码实现动态填充数据。两者的影响在于Rows属性中的下标使用不同，

即如果AllowUserToAddRows为true，在添加数据前，表格的行数已包含空行，因此，上面代码注释3前的语句应该写为：

dgv.Rows[count-1].Cells["Pos"].Value = coun;。

可以在表格的属性窗口中设置 Allow 开头的一组属性，实现在运行时是否可以允许用户直接在表格中添加行、删除行或调整列宽和行高等操作。通常表格仅仅用来显示数据，而添加和修改数据都是通过打开新窗口来实现，因此往往设置表格数据为只读，并在选择任意单元格时同时选中整行，这些属性既可以通过代码来设置，也可以在表格的属性窗口中设置。

应用实例 6.18 将菜品表 FoodItem 所有的信息，包括所属的菜系名称，在数据表格控件 DataGridView 中显示。其中菜系名称位于菜系表 FoodStyle 中。

新建项目，保存为 MyDataGridView。

界面布局

从工具箱的“数据”组中将 DataGridView 控件拖放到默认创建的窗体 Form1 中。并设置其 Dock 属性为 Fill 以填充窗体。

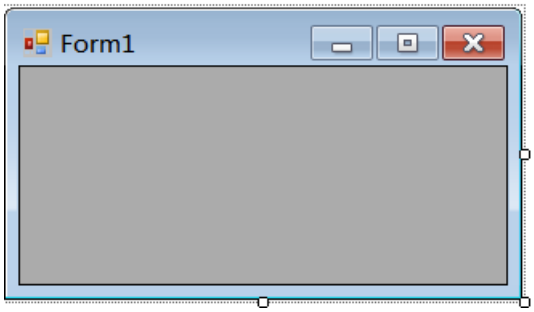

图 6.20　界面布局

添加一个实体类 FoodItem，用于保存从菜品表 FoodItem 读取的记录，FoodItem 的属性与 FoodItem 表的字段对应并同名，在 FoodItem 类中增加一个 StyleName 属性，对应菜系表 FoodStyle 的菜系名称。

程序代码 - FoodItem.cs 类文件

```
//省略自动生成的部分
public class FoodItem
{
    public int FoodID { get; set; }//菜品ID
    public string FoodName { get; set; }//菜品名称
    public int StyleID { get; set; }//菜系ID
    public string StyleName { get; set; }//菜系名称，来自FoodStyle表
    public float Price { get; set; }//价格
    public string ImagePath { get; set; }//图片名称
}
```

代码说明：为了使用方便，该类的属性与表的字段名同名。根据需要，可以增加实际表中不存在的属性，也可以去掉不需要的属性。

程序代码 - Form1.cs 类文件

```
//省略其他 using 声明部分
using System.Data.SqlClient;
namespace MyDataGridView
{
    public partial class Form1 : Form
    {
        public Form1()  {  InitializeComponent();  }//构造方法

        //1.自定义方法：创建表格，并设置相关属性
        void InitGridView()
        {
            dgv.Rows.Clear();     //2.清除数据行
            dgv.Columns.Clear();//3.清除数据列

            //4.创建表格列
            dgv.Columns.Add("Pos", "序号");//参数：列名，列标题
            dgv.Columns.Add("StyleID", "菜系ID");
            dgv.Columns.Add("StyleName", "菜系名称");
            dgv.Columns.Add("FoodID", "菜品ID");
            dgv.Columns.Add("FoodName", "菜品名称");
            dgv.Columns.Add("Price", "价格");
            dgv.Columns.Add("ImagePath", "图片位置");

            //5.设置列宽
            dgv.Columns[0].Width = 60;
            dgv.Columns[1].Width = 60;
            dgv.Columns[2].Width = 60;
            dgv.Columns[3].Width = 80;
            dgv.Columns[4].Width = 100;
            dgv.Columns[5].Width = 80;
            dgv.Columns[6].Width = 160;

            dgv.ReadOnly = true;                    //6.数据只读，不允许直接修改
            dgv.AllowUserToAddRows = true;         //7.不允许添加行
            dgv.AllowUserToDeleteRows = false;    //8.不允许删除行
            dgv.AllowUserToResizeRows = false;    //9.不允许调整行高
            dgv.RowHeadersVisible = false;         //10.隐藏行标头
            dgv.MultiSelect = false;               //11.不允许多行选择
            dgv.SelectionMode = DataGridViewSelectionMode.FullRowSelect;//12.选中整行
        }

        //13.自定义方法：将FoodItem类型参数包含的数据显示在表格中
        void FillGrid(FoodItem Food)
        {
```

```
    int count = dgv.Rows.Count;//14.当前行数
    dgv.Rows.Add();//15.添加空行

    dgv.Rows[count].Cells["Pos"].Value = count + 1;
    dgv.Rows[count].Cells["FoodID"].Value = Food.FoodID;
    dgv.Rows[count].Cells["FoodName"].Value = Food.FoodName;
    dgv.Rows[count].Cells["StyleID"].Value = Food.StyleID;
    dgv.Rows[count].Cells["StyleName"].Value = Food.StyleName;
    dgv.Rows[count].Cells["Price"].Value = Food.Price;
    dgv.Rows[count].Cells["ImagePath"].Value = Food.ImagePath;
}
```

//14.自定义方法：从FoodItem表和FoodStyle表中，读取所有的数据。

```
//每读取一行记录，作为参数，调用FillGrid方法，填充表格。
void GetFood()
{
        string connString = "Server=(local);DataBase=restaurant;
        Integrated Security=true";
        SqlConnection conn = new SqlConnection(connString);
        conn.Open();

        string sql = "select FoodItem.*,FoodStyle.* from FoodItem,FoodStyle
        Where FoodItem.StyleID=FoodStyle.StyleID";
        SqlCommand comm = new SqlCommand(sql, conn);
        SqlDataReader dr = comm.ExecuteReader();
        FoodItem Food = new FoodItem();

        while (dr.Read())
        {
            //读取并保存行数据
            Food.FoodID = int.Parse(dr["FoodID"].ToString());
            Food.FoodName = dr["FoodName"].ToString();
            Food.StyleID = int.Parse(dr["StyleID"].ToString());
            Food.StyleName = dr["StyleName"].ToString();
            Food.Price = float.Parse(dr["Price"].ToString());
            Food.ImagePath = dr["ImagePath"].ToString();
            FillGrid(Food);//自定义方法：填充记录到表格
        }
        dr.Close();conn.Close();
}
```

//15.窗体的Load事件处理过程，初始化表格，读取数据并显示到表格

```
private void Form1_Load(object sender, EventArgs e)
{
```

```
            InitGridView();
            GetFood();
        }

        //16.在表格中双击单元格时，取得当前选中行的数据
        private void dgv_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
        {
            if (dgv.CurrentRow == null) return;//17.判断当前有无选择行
            string selectInfo = "";
            selectInfo += dgv.CurrentRow.Cells["Pos"].Value + ",";
            selectInfo += dgv.CurrentRow.Cells["FoodID"].Value + ",";
            selectInfo += dgv.CurrentRow.Cells["FoodName"].Value + ",";
            selectInfo += dgv.CurrentRow.Cells["StyleID"].Value + ",";
            selectInfo += dgv.CurrentRow.Cells["StyleName"].Value + ",";
            selectInfo += dgv.CurrentRow.Cells["Price"].Value + ",";
            selectInfo += dgv.CurrentRow.Cells["ImagePath"].Value;
            this.Text = selectInfo;//18.测试选中行的数据，显示数据到标题栏
        }
    }//Form1类结束
}//命名空间结束
```

代码说明

(1) 注释1~注释12：自定义方法InitGridView，实现创建表格列，并设置各列的宽度，同时设置了表格的其他属性来限制用户操作，如表格数据只读、不允许用户在运行时添加数据、不允许用户删除行数据、不可调整行高、没有表格右侧的行标头、只能单行选择以及单击单元格时自动选择整行。这些属性都可以在表格的属性窗口中设置而不需要使用代码，这里只是给出使用代码实现的示例。

(2) 注释13：自定义方法FillGrid，在表格中创建新行，然后将数据依次填充到新行中的单元格中。

(3) 注释14：自定义方法GetFood，实现从数据库中的菜品表FoodItem和菜系表FoodStyle中读取数据。每读取一条记录，保存到Food对象中，并作为参数调用FillGrid方法来显示数据。

(4) 注释16：表格的CellDoubleClick事件处理过程，实现双击单元格时，取得选中行的数据，并连接成字符串显示到窗体标题栏中。

(5) 其余内容请参见代码中的注释。

运行结果

按 F5 键运行程序，运行结果如图 6.21 所示。

图6.21　运行结果

参考-典型案例 显示、添加、修改和删除数据。

本例实现对数据库 Restaurant 中的 FoodItem 表进行基本的数据操作，包括数据的显示、添加、修改和删除，运行效果如图 6.22 所示，具体实现功能如下：

(1) 在主窗体中，实现：①将数据库中 FoodItem 表的数据显示在 DataGridView 控件中；②删除在 DataGridView 控件选中的数据行，并更新到数据库。

(2) 在数据录入窗体中，将录入的数据显示到主窗体的 DataGridView 控件中，并更新到数据库。

(3) 在数据修改窗体中，显示并修改主窗体 DataGridView 控件选中的数据，同时将修改后的数据保存到数据库。

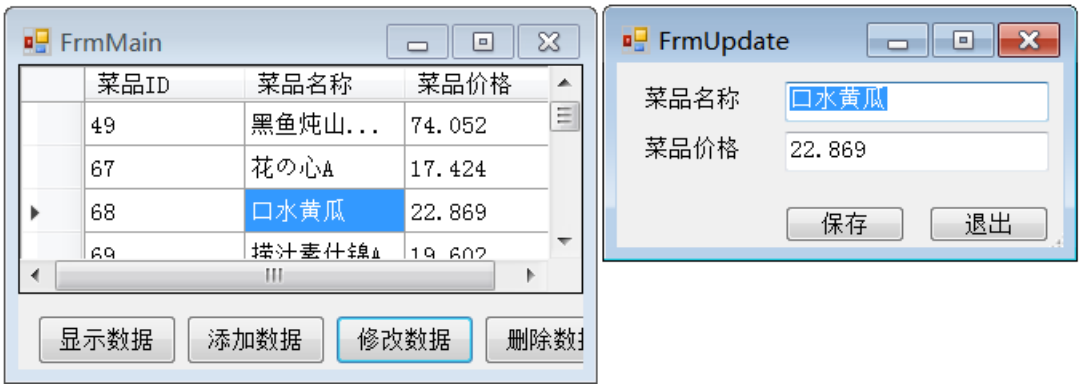

图 6.22　主窗体和数据修改窗体的运行效果

1. 案例分析

在项目实际开发过程中，要实现对一个数据表进行各种操作时，一般会创建一个主窗体，将数据以只读方式显示在表格中以供用户浏览，用户可以通过操作按钮删除数据行，而录入数据和修改数据都会在新窗口中进行，本例也遵循这一惯例来实现。

要快速从数据库读取表数据，可以使用 SqlDataReader 对象的方法；数据的显示可以使用表格控件 DataGridView；可以将在表格控件中选中的行数据作为一个整体(对象)传递到其他窗体，表格控件也可以作为对象传递到其他窗体，从而在其他窗体直接操作同一表格控件

的内容；添加和修改后的数据要更新到数据库，一般使用 SqlCommand 命令对象的 ExecuteNonQuery 方法来实现。

本例将综合前面已经介绍的内容，包括窗体对象的创建和显示，在窗体之间传递参数、对数据的查询和编辑等，以实际开发过程为顺序，分五大部分逐步介绍实现的过程。

2. 具体实现

创建项目，并保存为 MyDataTable。

(1) 主界面设计。

第 1 步：将默认创建的窗体文件 Form1.cs 重新命名为 FrmMain.cs，作为项目主窗体。

第 2 步：拖动 1 个 Panel 控件停靠在主窗体底部，设置其 Dock 属性为 Bottom；拖放 4 个按钮放到 Panel 控件中，分别重命名为 btShow、btAdd、btUpdate 和 btDel，并分别修改各个按钮的 Text 属性为“显示数据”、“添加数据”、“修改数据”和“删除数据”。

第 3 步：在工具箱的“数据”组中，拖放 1 个 DataGridView 控件到窗体，重命名为 dgv，设置其 Dcok 属性为 Fill，填满剩余的主窗体界面空间。

步骤 1~步骤 3 实现了如图 6.23 所示的主窗体的界面布局。

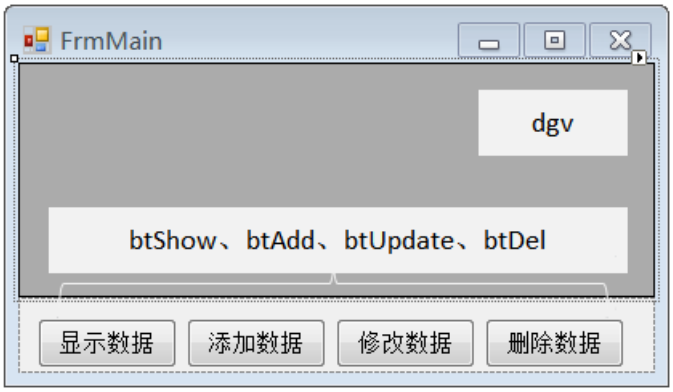

图 6.23　主窗体 FrmMain 的界面布局

第 4 步：为表格控件 dgv 添加 3 列数据列，列名分别为 FoodID、FoodName 和 Price，对应的列标题分别为“菜品 ID”、“菜品名称”和“菜品价格”。建立好的表格控件界面如图 6.24 所示。

图 6.24　包含 3 列的表格

第 5 步：设置表格控件 dgv 的属性，使其数据只读、用户不能直接在表格中添加行、修

改行和删除行，并且单击单元格时选中整行。在属性窗口中，设置以下表格的属性，初始化表格样式。

ReadOnly：True;//只读

AllowUserToAddRows：False;//不允许用户添加行

AllowUserToDeleteRows：False;//不允许用户删除

MultiSelect：False;//不允许多行选中

SelectionMode：FullRowSelect;//单击单元格时选中整行

在这里，表格控件 dgv 只起到显示数据的功能，而数据修改和添加功能将通过新建窗体来完成，目的是避免用户直接修改数据时出现“数据类型不匹配”的异常，而在新窗体可以方便对录入的数据进行验证。

(2) 添加类文件。

第 6 步：在项目中添加一个对应 FoodItem 表的实体类，类的属性名对应表的字段名，目的用于保存从数据表查询得到的记录。这样做的好处在于可以将一条记录作为一个整体对象，作为方法参数或在窗体之间传递，使得方法的参数更加简洁。FoodItem 类的代码如下：

```
public class FoodItem
{
    public int FoodID { get; set; }
    public string FoodName { get; set; }
    public float Price { get; set; }
}
```

第 7 步：添加一个 DB 类，定义一个公共静态字段 ConnString，初始化为连接字符串。静态公共字段相当于全局变量。由于在项目中多个窗体将使用到连接字符串，这样做可以避免重复定义连接字符串。DB 类代码如下：

```
public class DB
{
    public static string ConnString = "Server=.;DataBase=Restaurant;
    Integrated Security=true";
}
```

(3) 在主窗体中添加自定义方法 GetFood，实现将读取的数据显示到表格控件。

第8步：在窗体FrmMain代码文件中，添加一个自定义方法GetFood，用于读取数据库中FoodItem表的信息。每读取一条记录，都保存到FoodItem对象中，作为自定义方法FillGrid的参数。

注意：FrmMain代码文件必须声明对命名空间System.Data.SqlClient的引用。

```
void GetFood()
{
    SqlConnection conn = new SqlConnection(DB.ConnString);
    conn.Open();

    string sql = "Select * From FoodItem";
    SqlCommand comm = new SqlCommand(sql,conn);
    SqlDataReader dr = comm.ExecuteReader();
```

```
        while (dr.Read())
        {
            FoodItem fi = new FoodItem(); //保存行数据的对象di
            fi.FoodID = int.Parse( dr["FoodID"].ToString());
            fi.FoodName = dr["FoodName"].ToString();
            fi.Price = float.Parse(dr["Price"].ToString());
            FillGrid(fi);//作为行数据，填充表格控件
        }
        dr.Close();
        conn.Close();
    }
```

上面的代码用到了自定义方法 FiilGrid，该方法实现向表格控件 dgv 添加一行数据，代码如下：

```
    void FillGrid(FoodItem fi)
    {
        int count=dgv.Rows.Count;//取得当前行数
        dgv.Rows.Add();//添加空行
        //填充空行的单元格, Value为object类型，可以赋任何类型值
        dgv.Rows[count].Cells["FoodID"].Value = fi.FoodID;
        dgv.Rows[count].Cells["FoodName"].Value = fi.FoodName;
        dgv.Rows[count].Cells["Price"].Value = fi.Price;
    }
```

第 9 步：为“显示数据”按钮 btShow 编写 Click 事件处理过程，调用自定义方法显示数据。

```
    private void btShow_Click(object sender, EventArgs e)
    {
        dgv.Rows.Clear();//初始化表格控件
        GetFood();//自定义方法，读取表数据并显示到表格控件中
    }
```

(4) 添加新窗体 FrmAdd，实现数据录入并保存到数据库，同时更新主窗体表格。

第 10 步：为项目添加新窗体，保存为 FrmAdd。在该窗体中实现将录入的数据保存到数据库，并更新主窗体表格控件的数据。图 6.25 为 FrmAdd 的界面布局。

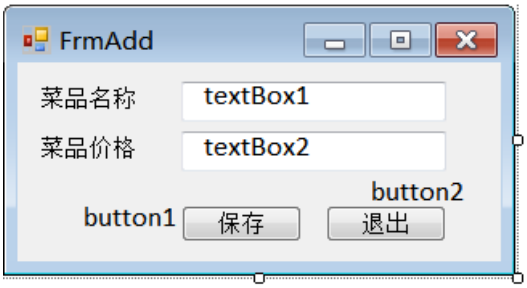

图 6.25　AddFrm 界面布局

在 FrmAdd 窗体代码文件中，添加 1 个 DataGridView 类型公共字段 dgv。

```
public DataGridView dgv;//用于保存从主窗体传递过来的表格对象
```

在主窗体单击“添加数据”按钮btAdd时，将创建FrmAdd对象。在显示FrmAdd窗体前，将主窗体表格控件dgv给FrmAddd对象的公共字段dgv赋值，两者指向都是同一个对象，这样在FrmAdd窗体中，dgv就可以直接操作主窗体的表格对象了。

注意：FrmAdd窗体代码文件必须声明对命名空间System.Data.SqlClient的引用。

第 11 步：为“保存”按钮 button1 添加事件处理过程并输入如下代码：

```
private void button1_Click(object sender, EventArgs e)
{
    FoodItem fi = new FoodItem();//用于暂存输入的数据。
    fi.FoodName = textBox1.Text;//取得菜品名称
    fi.Price = float.Parse(textBox2.Text);//取得菜品价格
    SaveFoodItem(fi);//保存到数据库
    UpdateGrid(fi); //更新主窗体数据的显示

    //保存后初始化
    textBox1.Text = ""
    textBox2.Text = "";
    textBox1.Focus();//取得输入焦点
}
```

代码中的自定义方法 SaveFoodItem 实现将界面输入的数据保存到数据库，方法 UpdateGrid 用于更新主窗体的表格数据，两个自定义方法的代码如下：

```
void SaveFoodItem(FoodItem fi)
{
    //保存到数据库
    SqlConnection conn = new SqlConnection(DB.ConnString);
    conn.Open();
    string sql = "Insert Into FoodItem(StyleID,FoodName,Price) ";
    sql += " Values(1,'" + fi.FoodName + "'," + fi.Price + ")";

    SqlCommand comm = new SqlCommand(sql, conn);
    comm.ExecuteNonQuery();
    conn.Close();
}
```

注意：添加数据到FoodItem表时，由于StyleID是外键，StyleID的值必须是存在于关联表FoodStyle表中。这里假定StyleID在FoodStyle中存在值为1的记录。

```
void UpdateGrid(FoodItem fi)
{
    //更新表格
    int count = dgv.Rows.Count;//当前表格行数
    dgv.Rows.Add();
```

```
        dgv.Rows[count].Cells["FoodName"].Value = fi.FoodName;
        dgv.Rows[count].Cells["Price"].Value = fi.Price;
    }
```

第12步：在主窗体FrmMain中，为“添加数据”按钮btAdd创建事件处理过程，实现创建FrmAdd窗体对象，并传递主窗体表格对象。在FrmAdd类中实现数据录入和保存，以及更新主窗体的表格数据。

主窗体中的“添加数据”按钮btAdd的事件处理过程的代码如下：

```
private void btAdd_Click(object sender, EventArgs e)
{
    FrmAdd frm = new FrmAdd();//创建窗体对象
    frm.dgv = this.dgv;//传递表格对象
    frm.ShowDialog();//打开窗体
}
```

(5) 添加新窗体 FrmUpdate，实现对在主窗体表格控件中选择的行数据进行操作。

第 13 步：为项目添加新窗体，保存为 FrmUpdate。在该窗体实现显示在主窗体表格控件中选中的行数据，以便核对修改，修改后保存到数据库，并更新主窗体表格控件选中行的数据。图 6.26 为 FrmUpdate 窗体的界面布局。

图 6.26　UpdateFrm 界面布局

在 UpdateFrm 窗体代码文件中，添加 1 个 DataGridView 类型公共字段。

```
public DataGridView dgv;
```

在主窗体单击“修改数据”按钮btUpdate时，将创建FrmUpdate对象，在显示FrmUpdate窗体前，将主窗体表格对象赋值给FrmUpdate窗体的公共字段dgv，两者指向同一个对象，那么在FrmUpdate窗体中，使用dgv就可以直接操作主窗体的表格对象了

注意：FrmUpdate窗体的代码文件必须声明对命名空间System.Data.SqlClient的引用。

在UpdateFrm窗体的代码文件中，添加一个FoodItem类型公共字段FoodItem，用于保存在主窗体表格控件中选中并传递过来的行数据。

```
public FoodItem fi;
```

在 UpdateFrm 窗体的 Load 事件处理过程，将传递过来的行数据显示到对应的文本框，代码如下：

```
private void UpdateFrm_Load(object sender, EventArgs e)
{
```

```
        textBox1.Text = fi.FoodName;
        textBox2.Text = fi.Price.ToString ();
    }
```

第 14 步：单击 FrmUpate 窗体上的“保存”按钮 button1 时，需要将修改后的数据保存到数据库，以及更新主窗体表格控件的数据显示，实现代码如下：

```
    private void button1_Click(object sender, EventArgs e)
    {
        fi.FoodName = textBox1.Text;
        fi.Price =float.Parse( textBox2.Text);
        SaveFoodItem(fi);//保存到数据库
        UpdateGrid(fi);//更新表格控件
        Close();//保存后关闭窗体
    }
```

上面代码中的自定义方法 SaveFoodItem 实现将修改后的的数据保存到数据库，自定义方法 UpdateGrid 用于更新主窗体表格控件的数据显示，两个方法的实现代码分别如下：

```
    void SaveFoodItem(FoodItem fi) //保存到数据库
    {
        SqlConnection conn = new SqlConnection(DB.ConnString);
        conn.Open();
        string sql = "Update FoodItem Set FoodName='" + fi.FoodName + "',Price=" + fi.Price;
        sql += " Where FoodID=" + fi.FoodID;
        SqlCommand comm = new SqlCommand(sql, conn);
        comm.ExecuteNonQuery();
        conn.Close();
    }
    void UpdateGrid(FoodItem fi) //更新表格
    {
        dgv.CurrentRow.Cells["FoodName"].Value = fi.FoodName;
        dgv.CurrentRow.Cells["Price"].Value = fi.Price;
    }
```

第15步：在主窗体FrmMain中，为“修改数据”按钮btUpdate创建事件处理过程，在该过程中首先判断是否选中行，如果没有，则退出过程；否则，创建FrmUpdate窗体对象，并传递表格对象以及选中行的数据，从而实现在FrmUpdate窗体更新数据。

主窗体中的“修改数据”按钮btUpdate的事件处理过程的代码如下：

```
    private void btUpdate_Click(object sender, EventArgs e)
    {
        if (dgv.CurrentRow == null) return;//如果没有选中行，终止执行
        FrmUpdate frm = new FrmUpdate();
        frm.dgv = this.dgv;//传递表格对象
        FoodItem fi = new FoodItem();//创建保存当前行数据的对象
        //由于单元格的Value属性是object类型，需要转换类型后再保存
        fi.FoodID = int.Parse(dgv.CurrentRow.Cells["FoodID"].Value.ToString());
```

```
        fi.FoodName = dgv.CurrentRow.Cells["FoodName"].Value.ToString();
        fi.Price = float.Parse(dgv.CurrentRow.Cells["Price"].Value.ToString());
        frm.fi = fi;//传递行数据
        frm.ShowDialog();
    }
```

第16步：在主窗体中实现数据删除。

单击主窗体的“删除数据”按钮btDel时，将根据选中行的主键列FoodID，实现从数据库中删除该行，并从表格移除选中行。

主窗体FrmMain中的“删除数据”按钮btDel的事件处理过程的代码如下：

```
private void btDel_Click(object sender, EventArgs e)
{
    if (dgv.CurrentRow == null) return;//如果没有选中行，终止执行
    int foodID = int.Parse(dgv.CurrentRow.Cells["FoodID"].Value.ToString());

    //根据选择的FoodID,从数据库中删除该条记录
    SqlConnection conn = new SqlConnection(DB.ConnString);
    conn.Open();

    string sql = "Delete From FoodItem Where FoodID="+foodID;
    SqlCommand comm = new SqlCommand(sql, conn);
    comm.ExecuteNonQuery();
    conn.Close();

    dgv.Rows.Remove(dgv.CurrentRow);//从表格中删除选中行
}
```

项目文件列表如图6.27所示。

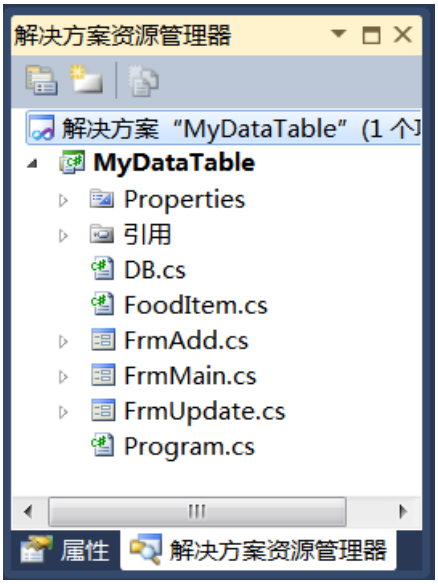

图6.27　项目文件

注意：在数据库Restauran中，FoodItem表中的FoodID、FoodName和Price列的数据不能为空，否则在读取数据时将发生异常，原因在于代码中没有对这些字段值进行验证，在进行类型转换时将出现错误；另外在FoodStyle表中，由于StyleID字段是FoodItem中具有约束关系的外键，因此必须至少存在StyleID值为1的行，否则，在执行数据添加操作时将出现“违反数据约束性”的异常。