

# **Pro ASP.NET 2.0 E-Commerce in C# 2005**



**Paul Sarknas**

## **Pro ASP.NET 2.0 E-Commerce in C# 2005**

**Copyright © 2006 by Paul Sarknas**

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-724-8

ISBN-10 (pbk): 1-59059-724-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Ewan Buckingham

Technical Reviewer: Fabio Claudio Ferracchiati

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade

Project Manager: Kylie Johnston

Copy Edit Manager: Nicole Flores

Copy Editor: Kim Wimpsett

Assistant Production Director: Kari Brooks-Copony

Production Editor: Kelly Winquist

Compositor/Artist: Kinetic Publishing Services, LLC

Proofreader: Lori Bring

Indexer: Brenda Miller

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail [info@apress.com](mailto:info@apress.com), or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Source Code/Download section.



# Designing the Database with SQL Server 2005

**Y**ou are finally moving along with the case study and beginning to deal with the engineering aspects of Little Italy Vineyards as opposed to the business side. This chapter will focus on building the database; this will be based on the model of the database and its objects that you developed in the previous chapter. In this chapter, I will discuss in detail how exactly you model and create the database; I'll provide step-by-step exercises along with a thorough explanation of the reasons for the modeling. I will also give some insight into why it is important to take the time in this early phase to pay special attention to designing the database with all its constraints and relationships.

This chapter will assume you have a moderate amount of experience with relational databases, specifically SQL Server 2005. Therefore, I will not explain what tables and stored procedures are and how they work, but I will explain what ones you will create and the reasons why you're creating them. If you find yourself needing to reference any fundamental information about SQL Server 2005, browse through the help files (more commonly known as Books Online), which have a wealth of information for you to digest.

The application for the Little Italy Vineyards case study will have several tables that you will create. This chapter will demonstrate how to create new tables, create new fields within the tables, and assign data types to the columns. Many of the tables when you are finished will have significant relationships to one another. Therefore, I'll also cover creating and managing the relationships in this chapter.

Please note that I'll show how to create the database, tables, and relationships through the SQL Server Management Studio interface. However, I will also supply the actual scripts to create these elements, so if you're more advanced, you can execute the scripts to create the database objects instead of using SQL Server Management Studio. Also, with this book's downloadable source code, the complete database script is available for your convenience. (See the introduction for more information about the code download.)

## Creating the Database

The first order of business is to create the new database within SQL Server 2005. The tool in which you manage all the activity in SQL Server 2005 is called SQL Server Management Studio.

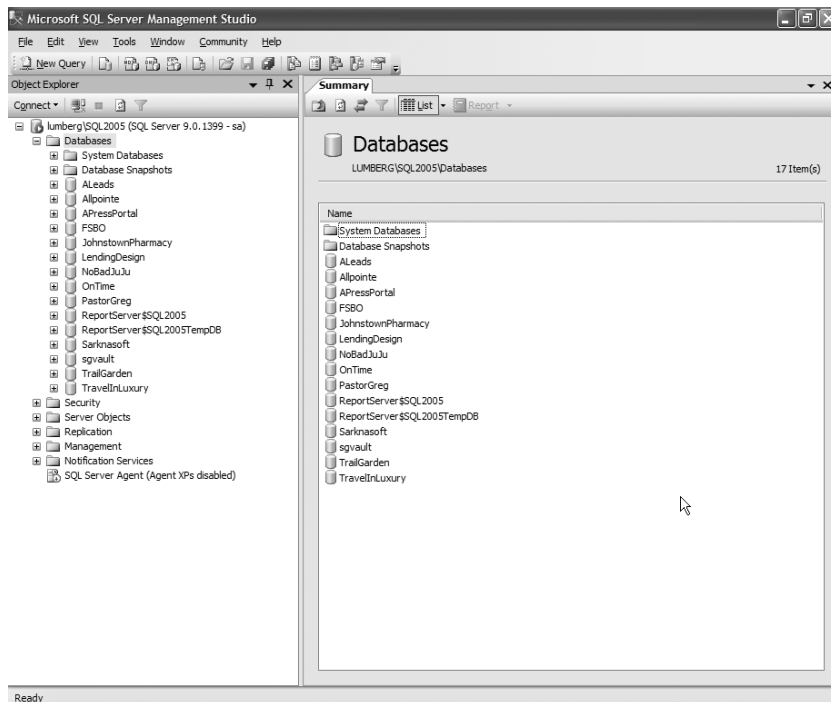
**Note** Although the examples in this chapter use SQL Server 2005, it is also possible to use SQL Server 2005 Express, which is free and does not require any licensing. Although SQL Server 2005 Express has its limitations, it might prove to be a better alternative for some projects.

The following exercise will walk you through creating the database for the Little Italy Vineyards case study.

### Exercise: Creating the Database

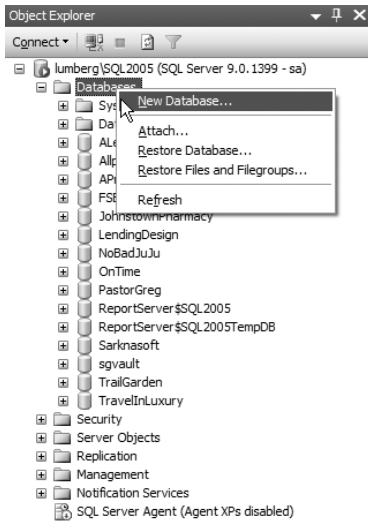
In the exercise, you'll create a new database within SQL Server 2005 from which you will build the associated tables and stored procedures:

1. Launch SQL Server Management Studio, and log in to the database engine that you will be utilizing for development within this project. After logging in to the database, you will see a tree from which you can expand the details of the database engine within Object Explorer, as shown in Figure 8-1. You can see that within my development environment, I have several databases that I have already created.



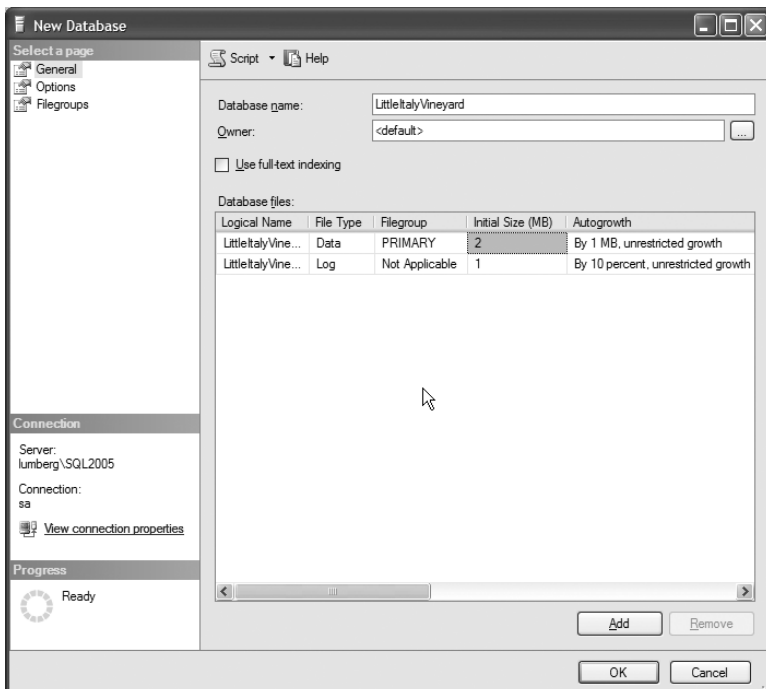
**Figure 8-1.** *Object Explorer*

2. Right-click the Databases directory listed directly below the main root in Object Explorer, and choose New Database, as shown in Figure 8-2.



**Figure 8-2.** *Creating a new database*

3. After right-clicking this menu item, you will see the dialog box shown in Figure 8-3 where you can specify the details of your new database. Enter the name of the database, **LittleItalyVineyard**, in the Database Name box. After entering this information, you can keep the remaining default features and save your addition by clicking the OK button.



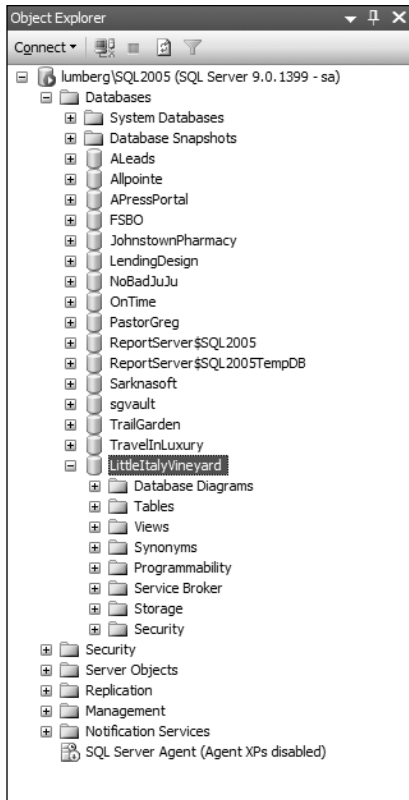
**Figure 8-3.** *Naming the new database*

If you would rather create the database by executing a script rather than using the graphical interface, you can execute the following script, which will provide you with the same result:

```
CREATE DATABASE LittleItalyVineyard
```

```
GO
```

After clicking the OK button or executing the script, you can use the new database to begin constructing the tables and stored procedures. At this point, you will see the new database in Object Explorer, as shown in Figure 8-4.



**Figure 8-4.** *Exploring the new database*

Once you have expanded the LittleItalyVineyard database, you can view the details of the database, which you'll explore in the following sections.

---

Since you have now created the database, you will proceed to the main focus of the chapter where you actually design the tables within the database.

## Creating the Tables

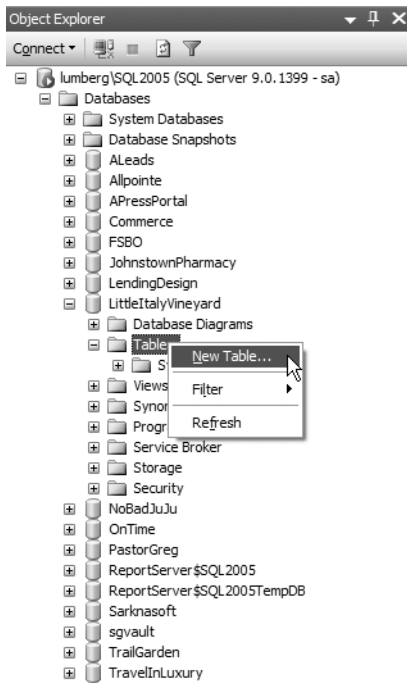
You'll now focus on adding the tables to the database. You have already modeled the objects within the system in the previous chapter, so you will use that information to create the tables. It is important to remember, however, that what you create and model in this chapter might not necessarily be the final outcome of what is needed.

In the following exercise, you will create the first table in the database. When creating subsequent tables for the database, use this exercise as the model. In the following sections, I'll discuss each of the tables in detail by providing an overall outline of each field in each table along with the data types and whether each field will allow null values. I'll also show a diagram of each table.

### Exercise: Creating a Table

As mentioned, you can use this exercise to create the first table and then use it as the base steps to create other tables. Without further ado, create the first table by following these steps:

1. After creating the database, expand the LittleItalyVineyard database within Object Explorer, right-click the Tables directory, and select New Table, as shown in Figure 8-5.



**Figure 8-5.** *Adding a new table*

After selecting this menu item, you will see the blank database shown in Figure 8-6 where you can enter the individual details. From this point within the exercise, you can begin constructing the individual fields.

	Column Name	Data Type	Allow Nulls
▶			<input type="checkbox"/>

Figure 8-6. Adding new fields

- 2. Begin entering the field names and the data types listed in Table 8-1 (in the next section), and indicate whether the field will allow null values. Figure 8-7 shows the final result of the table.

	Column Name	Data Type	Allow Nulls
▶	ProductID	int	<input type="checkbox"/>
	ProductCategoryID	int	<input type="checkbox"/>
	ProductName	nvarchar(50)	<input type="checkbox"/>
	ProductImageID	int	<input type="checkbox"/>
	Description	text	<input type="checkbox"/>
	Price	smallmoney	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 8-7. The new fields

- 3. After you have entered all the field names, defined the data types, and indicated whether each field will allow null values, you will set the primary key and any additional properties. Specifically, right-click the ProductID field, and select Set Primary Key, as shown in Figure 8-8.

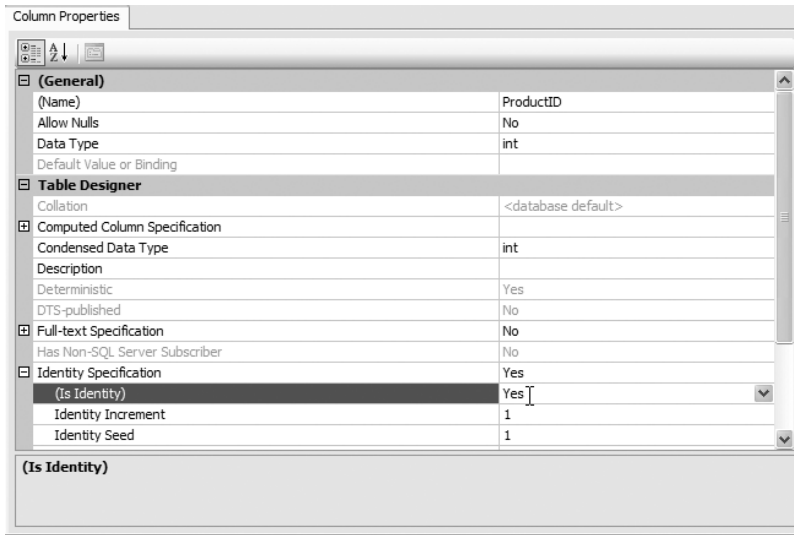
	Column Name	Data Type	Allow Nulls
▶	ProductID	int	<input type="checkbox"/>
	ProductCategoryID	int	<input type="checkbox"/>
	ProductName	nvarchar(50)	<input type="checkbox"/>
	ProductImageID	int	<input type="checkbox"/>
	Description	text	<input type="checkbox"/>
	Price	smallmoney	<input type="checkbox"/>
			<input type="checkbox"/>

- Set Primary Key
- Insert Column
- Delete Column
- Relationships...
- Indexes/Keys...
- Fulltext Index...
- XML Indexes...
- Check Constraints...
- Generate Change Script...

Figure 8-8. Setting the primary key

- 4. Now give the ProductID field a property of an autoincrement by clicking the option (Is Identity) and choosing Yes, as shown in Figure 8-9.





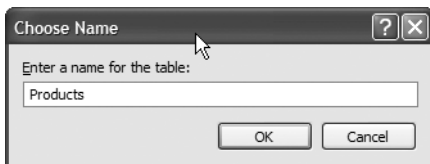
**Figure 8-9.** *Setting the identity property*

- Now you'll save the table for which you just entered the information. Specifically, click the Save icon, as shown in Figure 8-10.



**Figure 8-10.** *Saving the new table*

- When executing this action, you will be prompted to enter the name of the table. Enter **Products**, as shown in Figure 8-11.



**Figure 8-11.** *Naming the new table*

- Click the OK button to save and execute the actions.

If you want to create the table manually, you can execute the following script:

```
CREATE TABLE [Products]
(
    [ProductID] [int] IDENTITY(1,1) NOT NULL,
    [ProductCategoryID] [int] NOT NULL,
    [ProductName] [nvarchar](50) NOT NULL,
    [ProductImageID] [int] NOT NULL,
    [Description] [text] NOT NULL,
    [Price] [smallmoney] NOT NULL,
    CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
    [ProductID] ASC
)
)
```

You have created the first table within your database. As mentioned, follow this model when creating all the subsequent tables for your database.

---

## Products

The Products table was the first table you created because the system will focus on displaying and selling the individual products from the Little Italy Vineyards company. Table 8-1 shows the individual fields within the table.

**Table 8-1.** *Products Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	ProductID	int	No
		ProductCategoryID	int	No
		ProductName	nvarchar(50)	No
		ProductImageID	int	No
		Description	text	No
		Price	smallmoney	No

You'll notice in Table 8-1 that the first field, ProductID, has some properties that specify the field will be a primary key and will be an identity, or *autoincrement*, field. This means each time a new product is inserted into the table, the ProductID field will automatically give itself a value that is one greater than the previous entry in the table.

Now I'll discuss the individual fields and explain what their roles will be in the overall database, along with why you are choosing the specific data types. Figure 8-12 shows the fields.

Products			
	Column Name	Data Type	Allow Nulls
🔑	ProductID	int	<input type="checkbox"/>
	ProductCategoryID	int	<input type="checkbox"/>
	ProductName	nvarchar(50)	<input type="checkbox"/>
	ProductImageID	int	<input type="checkbox"/>
	Description	text	<input type="checkbox"/>
	Price	smallmoney	<input type="checkbox"/>
			<input type="checkbox"/>

**Figure 8-12.** *Products table*

## ProductID

The ProductID field provides the unique identifying number for a specific product being sold. This field is the primary key and has an autoincrement property; in addition, as a result of being a numeric value, its data type is an int, and the field will not allow any null values.

## ProductCategoryID

The ProductCategoryID field is a foreign key to the ProductCategory table. This will create a relationship from the Products table to the ProductCategory table (discussed in the “ProductCategory” section) in that each product will be associated to a specific category. As a result, the data type is an int, and the field will not allow any null values.

## ProductName

This is probably the most intuitive field in the table. The ProductName field simply holds the plain-text name of each product in the table. The data type is nvarchar(50), and this field will not allow any null values. If you find that your products have the tendency to have long names, you can increase the size of the nvarchar data type field; however, for the purposes of the case study, the names of the products will be on the shorter side.

## ProductImageID

The ProductImageID field is a foreign key to the ProductImages table. I will explain more about the ProductImages table later in the “ProductImages” section; however, for now just know that all products will have an associated image to display. The images within the system will be stored in the database, as opposed to having them in the file system and providing the path to the images. This field will create the association to the image from the ProductImages table. As a result, it is an int data type and will not allow null values.

## Description

Another quite obvious field in the Products table is the Description field. This, as the name implies, provides the description of the product. These descriptions can be quite lengthy at times, so to allow for this, this field will use the text data type and not allow null values.

Price

The Price field holds the monetary value of the cost of each product in the table. Since it will be a currency value, the data type will be a smallmoney type, and the field not allow null values.

At this point, you might be asking yourself, why not have a field for shipping cost and sales tax when charging sales tax for the products sold? You will be dealing with these values; however, they are considered to be aggregate values, and it is a good practice to not store aggregate values within your database schema. Instead, these values will be automatically calculated.

ProductCategory

Continuing along with products, the next table you'll create should be the ProductCategory table. This table will be a fairly simple table with only a few fields because this table will be what is known as a *lookup table*. In other words, it will give specific category names to all the products while associating them to the Products table with a unique ID. Table 8-2 shows the specifics of the ProductCategory table.

Table 8-2. ProductCategory Table

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	ProductCategoryID	int	No
		ProductCategoryName	text	No

In the following sections, I'll discuss the individual fields of the ProductCategory table. Figure 8-13 shows the fields.

ProductCategory			
	Column Name	Data Type	Allow Nulls
PK	ProductCategoryID	int	<input type="checkbox"/>
	ProductCategoryName	text	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 8-13. ProductCategory table

ProductCategoryID

The ProductCategoryID field is the unique identifier in the table along with being the primary key. As a result, it will have a data type of int and not allow any null values.

ProductCategoryName

The ProductCategoryName field simply gives the text description of what the category is. This field can be lengthy, or it can be relatively brief. To allow for maximum scalability, you will use the data type of text and not allow null values.

## Table Script

The following script will create a new table named ProductCategory:

```
CREATE TABLE [ProductCategory]
(
    [ProductCategoryID] [int] IDENTITY(1,1) NOT NULL,
    [ProductCategoryName] [text] NOT NULL,
    CONSTRAINT [PK_ProductCategory] PRIMARY KEY CLUSTERED
    (
        [ProductCategoryID] ASC
    )
)
```

## ProductImages

The ProductImages table is also brief and contains only a few fields. This table will primarily contain binary data for the images that are associated with the products. The natural question to ask is, why not simply have an image field with the Products table itself? Well, having a separate table for the image data will provide greater performance when querying the data from a dedicated table containing the binary images. Table 8-3 shows the specifics of the ProductImages table.

**Table 8-3.** *ProductImages Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	ProductImageID	int	No
		ProductImage	image	No

In the following sections, I'll discuss the individual fields of the ProductImages table. Figure 8-14 shows the fields.

	Column Name	Data Type	Allow Nulls
PK	ProductImageID	int	<input type="checkbox"/>
	ProductImage	image	<input type="checkbox"/>

**Figure 8-14.** *ProductImages table*

## ProductImageID

The ProductImageID field is the primary key of the table and has an autoincrement property. Therefore, keeping with the consistency of the other tables, the data type will be an int, and no null values will be allowed.

ProductImage

The ProductImage field is the only other field in the table; however, it has some unique characteristics compared to some of the others to this point. This field will contain the binary information for the image that is associated with the product. As a result of the information being in its binary form, the data type will be image, and no null values will be allowed.

Table Script

The following script will create the ProductImages table:

```
CREATE TABLE [ProductImages]
(
    [ProductImageID] [int] IDENTITY(1,1) NOT NULL,
    [ProductImage] [image] NOT NULL,
    CONSTRAINT [PK_ProductImages] PRIMARY KEY CLUSTERED
    (
        [ProductImageID] ASC
    )
)
```

Orders

The Orders table will be a major portion of the overall database. This table contains all the information about the items a customer intends to purchase. Table 8-4 shows the specifics of the Orders table.

Table 8-4. Orders Table

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	OrderID	int	No
		TransactionID	nvarchar(50)	No
		EndUserID	int	No
		OrderStatusID	int	No
		OrderDate	smalldatetime	No
		ShipDate	smalldatetime	Yes
		TrackingNumber	nvarchar(50)	Yes

In the following sections, I'll discuss the individual fields of the Orders table. Figure 8-15 shows the fields.

	Column Name	Data Type	Allow Nulls
PK	OrderID	int	<input type="checkbox"/>
	TransactionID	nvarchar(50)	<input type="checkbox"/>
	EndUserID	int	<input type="checkbox"/>
	OrderStatusID	int	<input type="checkbox"/>
	OrderDate	smalldatetime	<input type="checkbox"/>
	ShipDate	smalldatetime	<input checked="" type="checkbox"/>
	TrackingNumber	nvarchar(50)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

**Figure 8-15.** *Orders table*

### OrderID

The OrderID field is the same as the other primary key fields in the other tables thus far. It will also have an autoincrement property along with a data type of int, and no null values will be permitted.

### TransactionID

The TransactionID field is a string of characters that is returned from PayPal when processing the payment. This unique ID allows you to refer to the transaction that has been processed with PayPal and even issue a refund if necessary.

### EndUserID

The EndUserID field simply provides a foreign key of the unique identification of the customer who has registered an account with the company and provided their contact information. This ID will reside in the EndUser table, which will be discussed later in the “EndUser” section. Lastly, the EndUserID field will have a data type of int and not allow any null values.

### OrderStatusID

The OrderStatusID field refers to the OrderStatus table and corresponds to the appropriate status type of the order. The default value for this column is 1, which can be added to the column properties by entering a value of 1 for the Default Value or Binding property.

### OrderDate

The OrderDate field simply contains the date when the order is created. To provide additional simplicity, this field has a default value that is equal to the current date. Therefore, when a new order is inserted into the table, the code or stored procedure will not have to specify that OrderDate field, and as a result, the current date will be taken from what’s found on the server on which SQL Server 2005 is running. You can do this by specifying the GetDate() function in the Default Value or Binding column property. As a result, the data type used will be a small-datetime and not allow null values.

## ShipDate

The ShipDate field holds the date of when the order is shipped to the customer. It primarily is for historic purposes when either the administrator or the customer needs to find out when their order has been shipped so they can anticipate when it will actually be delivered. This field actually allows null values because you will not know when the order will ship at the time the order is created. When this information is prepared, the table will need to be updated for this date.

## TrackingNumber

The TrackingNumber field contains the identifying field that is provided by the shipping company. This will usually be a long number that also includes alpha characters, so to provide for this, the data type is an nvarchar(50), and the field will also allow null values. This is because you will not know the tracking number until the shipping company has arranged to pick up the order and has provided the company with this information. At that time, the tracking number will be updated in the table.

## Table Script

The following script will create the Orders table:

```
CREATE TABLE [Orders]
(
    [OrderID] [int] IDENTITY(1,1) NOT NULL,
    [TransactionID] [nvarchar](50) NOT NULL,
    [EndUserID] [int] NOT NULL,
    [OrderStatusID] [int] NOT NULL DEFAULT ((1)),
    [OrderDate] [smalldatetime] NOT NULL DEFAULT (getdate()),
    [ShipDate] [smalldatetime] NULL,
    [TrackingNumber] [nvarchar](50) NULL,
    CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
    (
        [OrderID] ASC
    )
)
```

## OrderDetails

The OrderDetails table provides all the information regarding the details of a specific order. Each order that a customer enters has the ability to have as many individual items as they want. In other words, the customer can purchase one product or can order 25 different products—it makes no difference. Table 8-5 shows the specifics of the OrderDetails table.



**Table 8-5.** *OrderDetails Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	OrderDetailID	int	No
		OrderID	int	No
		ProductID	int	No
		Quantity	int	No

Figure 8-16 shows the fields of the OrderDetails table.

OrderDetails			
	Column Name	Data Type	Allow Nulls
?	OrderDetailID	int	<input type="checkbox"/>
	OrderID	int	<input type="checkbox"/>
	ProductID	int	<input type="checkbox"/>
	Quantity	int	<input type="checkbox"/>
			<input type="checkbox"/>

**Figure 8-16.** *OrderDetails table*

### OrderDetailID

The OrderDetailID field is the primary key and is an autoincrement that will be the unique identifier for the associated record within the OrderDetails table.

### OrderID

The OrderID field is the foreign key that relates to the Orders table and associates the individual order detail to the overall order. This field will have a data type of int and will not allow null values.

### ProductID

The ProductID field is the foreign key that relates to the Products table to specify what product is being added to be eventually purchased. The data type will be int, and the field will not allow null values.

### Quantity

The Quantity field is exactly what its name indicates. It indicates how many products are being requested for the purchase. It will have a data type of int and not allow any null values, because at least one product will need to be indicated for the order.

Table Script

The following script will create the OrderDetails table:

```
CREATE TABLE [OrderDetails]
(
    [OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
    [OrderID] [int] NOT NULL,
    [ProductID] [int] NOT NULL,
    [Quantity] [int] NOT NULL,
    CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
    (
        [OrderDetailID] ASC
    )
)
```

OrderStatus

The OrderStatus table provides the associated status names for any order that is placed within the system. It will have two separate columns: one for the associated ID and the other for the name of the status. Table 8-6 shows the specifics of the OrderStatus table.

Table 8-6. OrderStatus Table

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	OrderStatusID	int	No
		OrderStatusName	nvarchar(50)	No

Figure 8-17 shows the fields of the OrderStatus table.

OrderStatus			
	Column Name	Data Type	Allow Nulls
PK	OrderStatusID	int	<input type="checkbox"/>
	OrderStatusName	nvarchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 8-17. OrderStatus table

OrderStatusID

The OrderStatusID field serves as the primary key field and has an autoincrement property. This ID will relate any order’s status to other tables within the database.

OrderStatusName

The OrderStatusName field is the associated text or string name to that of the foreign key field, OrderStatusID, that will relate to the Orders table. Setting the structure up in this way will allow for maximum scalability of the different status names. Lastly, the data type is an nvarchar(50), and no null values will be allowed.

## Table Script

The following script will create the OrderStatus table:

```
CREATE TABLE [OrderStatus]
(
    [OrderStatusID] [int] IDENTITY(1,1) NOT NULL,
    [OrderStatusName] [nvarchar](50) NOT NULL,
    CONSTRAINT [PK_OrderStatus] PRIMARY KEY CLUSTERED
    (
        [OrderStatusID] ASC
    )
)
```

## EndUser

The EndUser table contains all the information regarding the users who will be interacting within the system. The users will range from administrators to customers, but regardless, to keep the database and its information normalized, all this information will be contained in a single table. Table 8-7 shows the specifics of the EndUser table.

**Table 8-7.** *EndUser Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	EndUserID	int	No
		EndUserTypeID	int	No
		FirstName	nvarchar(50)	No
		LastName	nvarchar(50)	No
		AddressID	int	No
		ContactInformationID	int	No
		Password	nvarchar(50)	No
		IsSubscribed	bit	No

I'll discuss the fields of the EndUser table in the following sections; Figure 8-18 shows them.

Column Name	Data Type	Allow Nulls
EndUserID	int	<input type="checkbox"/>
EndUserTypeID	int	<input type="checkbox"/>
FirstName	nvarchar(50)	<input type="checkbox"/>
LastName	nvarchar(50)	<input type="checkbox"/>
AddressID	int	<input type="checkbox"/>
ContactInformationID	int	<input type="checkbox"/>
Password	nvarchar(50)	<input type="checkbox"/>
IsSubscribed	bit	<input type="checkbox"/>

**Figure 8-18.** *EndUser table*

### **EndUserID**

The EndUserID field serves as the primary key field and has an autoincrement property. This ID will relate any user to other tables within the database.

### **EndUserTypeID**

The EndUserTypeID field is a foreign key field that will relate to the EndUserType table, which will specify what type of user the specific user is. Having the structure set up in this way will allow for maximum scalability of the different types of users. Lastly, the data type is an int, and no null values will be allowed.

### **FirstName**

The FirstName field is exactly what its name indicates. It will contain the information of the user's first name. Therefore, the data type will be an nvarchar(50), and no null values will be permitted.

### **LastName**

The LastName field is exactly what its name indicates. It will contain the information of the user's last name. Therefore, the data type will be an nvarchar(50), and no null values will be permitted.

### **AddressID**

The AddressID field will contain the foreign key related to the Address table in the database. I will discuss the Address table in the "Address" section. The data type is an int, and no null values will be allowed.

### **ContactInformationID**

The ContactInformationID field contains the foreign key related to the ContactInformation table within the database. I'll discuss the ContactInformation table in the "ContactInformation" section. Therefore, the data type will be an int, and no null values will be allowed.

### **Password**

The Password field contains the password that the user supplies to get access to their account. Their password will be encrypted within the database to provide additional security. As a result, the data type used will be an nvarchar(50) and will not allow null values.

### **IsSubscribed**

The IsSubscribed field provides for the value that is either a 0 or a 1. This value will be set when a customer or user registers for a new account; they can elect to subscribe to the newsletter.

## Table Script

The following script will create the EndUser table:

```
CREATE TABLE [EndUser]
(
    [EndUserID] [int] IDENTITY(1,1) NOT NULL,
    [EndUserID] [int] NOT NULL,
    [FirstName] [nvarchar](50) NOT NULL,
    [LastName] [nvarchar](50) NOT NULL,
    [AddressID] [int] NOT NULL,
    [ContactInformationID] [int] NOT NULL,
    [Password] [nvarchar](50) NOT NULL,
    [IsSubscribed] [bit] NOT NULL,
    CONSTRAINT [PK_EndUser] PRIMARY KEY CLUSTERED
    (
        [EndUserID] ASC
    )
)
```

## EndUserType

The EndUserType table will be a brief lookup table to specify the different roles a user can have and to provide different categorizations. Table 8-8 shows the specifics of the EndUserType table.

**Table 8-8.** *EndUserType Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	EndUserTypeID	int	No
		TypeName	nvarchar(50)	No

I'll discuss the fields of the EndUserType table in the following sections; Figure 8-19 shows them.

Column Name	Data Type	Allow Nulls
EndUserTypeID	int	<input type="checkbox"/>
TypeName	nvarchar(50)	<input type="checkbox"/>

**Figure 8-19.** *EndUserType table*

### EndUserTypeID

The EndUserTypeID field serves as the primary key field and has an autoincrement property. Its data type will be an int, and the field will not allow null values.

**TypeName**

The TypeName field provides the text description of what the EndUserID will be. This descriptive field will usually have the name Administrator, Customer, or perhaps Vendor.

**Table Script**

The following script will create the EndUserType table:

```
CREATE TABLE [EndUserType]
(
    [EndUserID] [int] IDENTITY(1,1) NOT NULL,
    [TypeName] [nvarchar](50) NOT NULL,
    CONSTRAINT [PK_EndUserType] PRIMARY KEY CLUSTERED
    (
        [EndUserID] ASC
    )
)
```

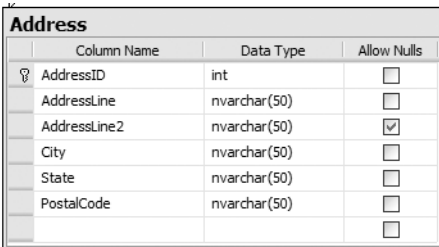
**Address**

The Address table contains all the address information. It will contain address information only and can be used for users or any other part of the system or database that needs to include address information. For the purposes of the case study, it will mostly be used for the EndUser table. This structure is a technique used to further normalize the data within the schema of the database. Table 8-9 shows the specifics of the Address table.

**Table 8-9.** *Address Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	AddressID	int	No
		AddressLine	nvarchar(50)	No
		AddressLine2	nvarchar(50)	Yes
		City	nvarchar(50)	No
		State	nvarchar(50)	No
		PostalCode	nvarchar(50)	No

I'll discuss the fields of the Address table in the following sections; Figure 8-20 shows them.



	Column Name	Data Type	Allow Nulls
PK	AddressID	int	<input type="checkbox"/>
	AddressLine	nvarchar(50)	<input type="checkbox"/>
	AddressLine2	nvarchar(50)	<input checked="" type="checkbox"/>
	City	nvarchar(50)	<input type="checkbox"/>
	State	nvarchar(50)	<input type="checkbox"/>
	PostalCode	nvarchar(50)	<input type="checkbox"/>

**Figure 8-20.** *Address table*

### AddressID

The AddressID field is the primary key and has an autoincrement property. Therefore, the data type will be int, and the field will not allow null values.

### AddressLine

The AddressLine field contains the information for the street number and the street name of an entity's address. For instance, "105 Main St." could be a value that is found in the table. Therefore, the data type is an nvarchar(50) and does not allow null values.

### AddressLine2

The AddressLine2 field contains the information for addresses that sometimes have additional information. For instance, an address might be "100 South Main Blvd." but might also need to specify a suite or maybe even an apartment number. Therefore, information found in this field can often resemble that of "Suite 200" or "Apartment Number 4D." As a result, the data type is an nvarchar(50), and since only some addresses will have this additional information, null values will be allowed.

### City

The City field specifies the name of the city within the full address. This will contain the full text, such as "Pittsburgh" or "Tampa Bay." As a result, the data type will be an nvarchar(50), and the field will not allow null values.

### State

The State field specifies the name of the state within the full address. This will contain the full text, such as "Pennsylvania" or "Florida." As a result, the data type will be an nvarchar(50), and the field will not allow null values.

### PostalCode

The PostalCode field specifies the value of the postal code (or the ZIP code) for the full address. An example is 15222 or 15282; therefore, this field will have a data type of nvarchar(50) and not allow null values.

Table Script

The following script will create the Address table:

```
CREATE TABLE [Address]
(
    [AddressID] [int] IDENTITY(1,1) NOT NULL,
    [AddressLine] [nvarchar](50) NOT NULL,
    [AddressLine2] [nvarchar](50) NULL,
    [City] [nvarchar](50) NOT NULL,
    [State] [nvarchar](50) NOT NULL,
    [PostalCode] [nvarchar](50) NOT NULL,
    CONSTRAINT [PK_Address] PRIMARY KEY CLUSTERED
    (
        [AddressID] ASC
    )
)
```

ContactInformation

The ContactInformation table is similar to that of the Address table in the database. However, instead of containing address information for a specific entity, it specifies contact information for an entity to keep with the normalization technique mentioned within the Address table description. By utilizing this technique, all contact information can be in one table and be associated with any type of entity. For the purposes of the case study, the users are the only entities that have related contact information. Table 8-10 shows the specifics of the ContactInformation table.

Table 8-10. ContactInformation Table

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	ContactInformationID	int	No
		Phone	nvarchar(50)	Yes
		Phone2	nvarchar(50)	Yes
		Fax	nvarchar(50)	Yes
		Email	nvarchar(50)	No

I'll discuss the fields of the ContactInformation table in the following sections; Figure 8-21 shows them.



ContactInformation			
	Column Name	Data Type	Allow Nulls
?	ContactInformationID	int	<input type="checkbox"/>
	Phone	nvarchar(50)	<input checked="" type="checkbox"/>
	Phone2	nvarchar(50)	<input checked="" type="checkbox"/>
	Fax	nvarchar(50)	<input checked="" type="checkbox"/>
	Email	nvarchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

**Figure 8-21.** *ContactInformation table*

### ContactInformationID

The ContactInformationID field serves as the primary key and has the autoincrement property. Therefore, it will have a data type of int and not allow null values.

### Phone

The Phone field contains the raw numbers of the primary phone number of the overall contact information entity. There will be no formatting of the phone number because any formatting will be performed on the presentation layer side of the system. It will have a data type of nvarchar(50) and allow null values.

### Phone2

The Phone2 field contains the raw numbers of a secondary phone number of the overall contact information entity. Perhaps this is a cellular or mobile phone number. However, not all entities will have a secondary number; therefore, it will allow null values and have a data type of nvarchar(50).

### Fax

The Fax field specifies the raw numbers of an entity's fax. Not every entity will have an associated fax; therefore, this field will allow null values and have a data type of nvarchar(50).

### Email

The Email field contains the e-mail address of the entity's contact information. It will show the e-mail address in plain text; therefore, the data type will be an nvarchar(50), and no null values will be permitted.

### Table Script

The following script will create the ContactInformation table:

```
CREATE TABLE [ContactInformation]
(
    [ContactInformationID] [int] IDENTITY(1,1) NOT NULL,
    [Phone] [nvarchar](50) NULL,
```

```

[Phone2] [nvarchar](50) NULL,
[Fax] [nvarchar](50) NULL,
[Email] [nvarchar](50) NOT NULL,
CONSTRAINT [PK_ContactInformation] PRIMARY KEY CLUSTERED
(
    [ContactInformationID] ASC
)
)
```

## ShoppingCart

The ShoppingCart table contains all the information from when customers are browsing through the product catalog and then adding a specific item to the shopping cart to eventually purchase. Table 8-11 shows the specifics of the ShoppingCart table.

**Table 8-11.** *ShoppingCart Table*

Autoincrement?	Primary Key?	Field Name	Data Type	Allow Null?
Yes	Yes	ShoppingCartID	int	No
		CartGUID	Nvarhcar(50)	No
		Quantity	int	No
		ProductID	int	No
		DateCreated	smalldatetime	No

I'll discuss the fields of the ShoppingCart table in the following sections; Figure 8-22 shows them.

ShoppingCart			
	Column Name	Data Type	Allow Nulls
?	ShoppingCartID	int	<input type="checkbox"/>
	CartGUID	nvarchar(50)	<input type="checkbox"/>
	Quantity	int	<input type="checkbox"/>
	ProductID	int	<input type="checkbox"/>
	DateCreated	smalldatetime	<input type="checkbox"/>
			<input type="checkbox"/>

**Figure 8-22.** *ShoppingCart table*

### ShoppingCartID

The ShoppingCartID field is the primary key of the table and has an autoincrement property. Therefore, the data type will be an int, and no null values will be allowed.

### CartGuid

The CartGuid field is a global unique identifier (GUID) that is created from the ASP.NET code and then passed along to be inserted into the database. This identifier will allow for a customer to have all the items that they have placed within the shopping cart be identified as theirs.

## Quantity

The Quantity field holds the value of how many of the individual products the customer would like when adding them to the shopping cart. As a result of the values being a single number, the data type will be an int, and the field not allow null values, because at least a value of 1 must be entered.

## ProductID

The ProductID field is a foreign key to the Products table to specify and relate a specific product item to the cart. This will have a data type of an int and not allow null values.

## DateCreated

The DateCreated field provides a date stamp of when the user creates the shopping cart by adding products. The value of the date will be generated by the GetDate() function placed within the Default Value property. This will have a data type of smalldatetime and will have a default value of using the current date, which will be extracted from the server's date on which SQL Server 2005 is installed and running. This field will be important when processing abandoned shopping carts, which will be explained in greater detail in Chapter 17.

## Table Script

The following script will create the ShoppingCart table:

```
CREATE TABLE [ShoppingCart]
(
    [ShoppingCartID] [int] IDENTITY(1,1) NOT NULL,
    [CartGUID] [nvarchar](50) NOT NULL,
    [Quantity] [int] NOT NULL,
    [ProductID] [int] NOT NULL,
    [DateCreated] [smalldatetime] NOT NULL DEFAULT (getdate()),
    CONSTRAINT [PK_ShoppingCart] PRIMARY KEY CLUSTERED
    (
        [ShoppingCartID] ASC
    )
)
```

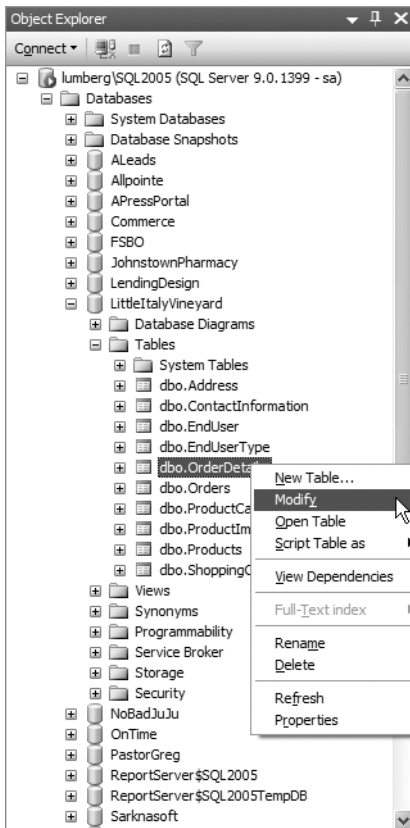
## Creating the Relationships

As mentioned earlier within the chapter, several of the tables will have distinct relationships with each other. At this point, you have created each of the necessary tables along with their respective data types and associated properties. Similar to the methodology earlier explained when creating the tables, you will follow a single exercise to create a single relationship between two tables. You can then use this as a model to follow when creating the other relationships.

## Exercise: Creating a Relationship

This exercise will outline how to create a relationship between two tables. A table might have a single relationship or multiple relationships. In either case, utilize this exercise to model all the relationships for the tables within the database:

1. Within Object Explorer, expand the Tables node, and choose the table from which you want to create a relationship. For this exercise, you will be dealing with the OrderDetails table. Therefore, right-click the OrderDetails table, and choose Modify, as shown in Figure 8-23.



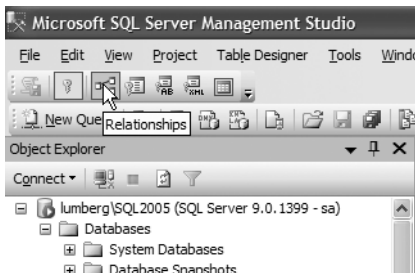
**Figure 8-23.** *Modifying a table*

After choosing Modify, you will see the table details, as shown in Figure 8-24.

Column Name	Data Type	Allow Nulls
OrderDetailID	int	<input type="checkbox"/>
OrderID	int	<input type="checkbox"/>
ProductID	int	<input type="checkbox"/>
Quantity	int	<input type="checkbox"/>
		<input type="checkbox"/>

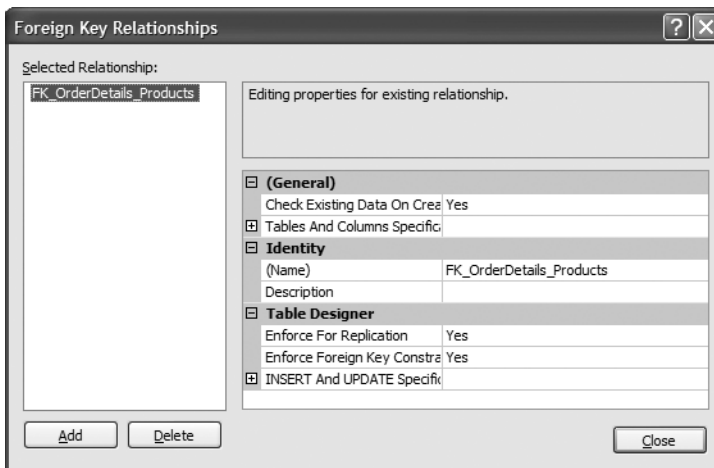
**Figure 8-24.** *The table columns*

2. From here, click the Relationships icon to view the current relationships, as shown in Figure 8-25.



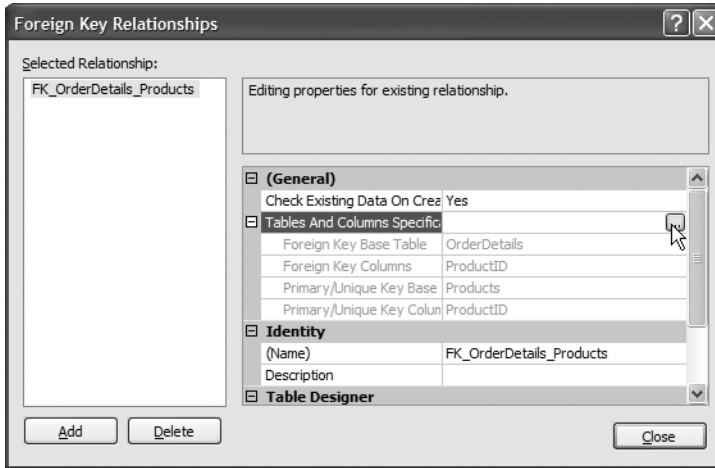
**Figure 8-25.** *The relationships*

3. After clicking the Relationships icon, you will see the dialog box shown in Figure 8-26 where you can click the Add button.



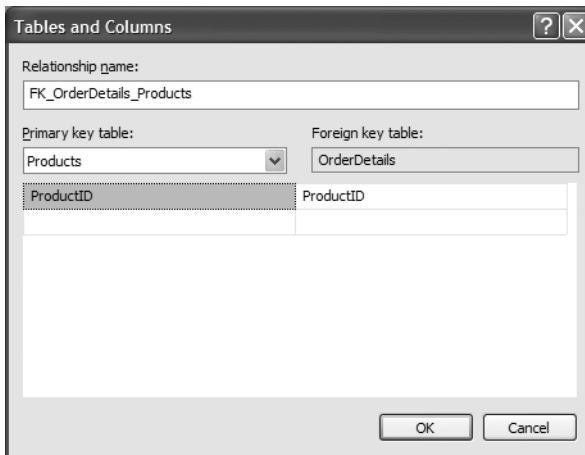
**Figure 8-26.** *Foreign key relationships*

4. In the Foreign Key Relationship dialog box, expand the Tables and Columns Specification property, as shown in Figure 8-27.



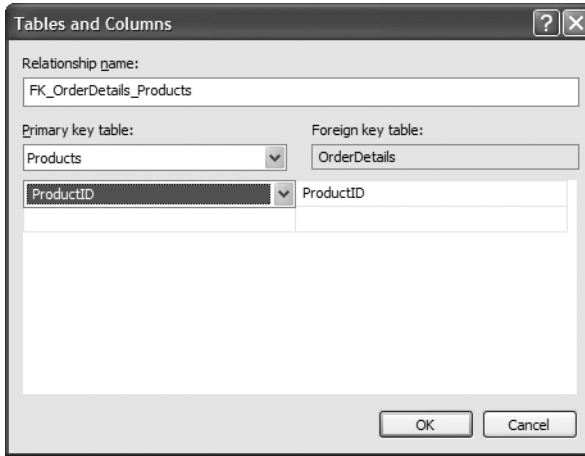
**Figure 8-27.** Adding a relationship

5. After clicking the ellipsis, you will be presented with the Tables and Columns dialog box, as shown in Figure 8-28.



**Figure 8-28.** Setting the relationship

6. From the drop-down list on the left, choose the Products table, and then select the ProductID field from within the OrderDetails table, as shown in Figure 8-29.



**Figure 8-29.** *Saving the relationship*

7. Upon ensuring that both the ProductID selections are made in both the columns, click the OK button to execute the changes. You have created the first relationship.

As an alternative to creating the relationship using the graphical user interface, you can run the following script to create the relationship:

```
ALTER TABLE [OrderDetails] WITH CHECK ADD CONSTRAINT
[FK_OrderDetails_Products] FOREIGN KEY([ProductID])
REFERENCES [Products] ([ProductID])
```

Follow this model for creating all the subsequent relationships for the other tables within the database that I'll outline in the remainder of the chapter.

## OrderDetails

The OrderDetails table has two relationships. These relationships are with OrderID and ProductID from the Orders and Products tables, respectively. Table 8-12 shows the specifics of the OrderDetails relationships.

**Table 8-12.** *OrderDetails Relationships*

Constraint Name	Foreign Key	Reference Table	Reference Field
FK_OrderDetails_Orders	OrderID	Orders	OrderID
FK_OrderDetails_Products	ProductID	Products	ProductID

### FK\_OrderDetails\_Orders

The first constraint allows only an existing OrderID from the Orders table within the OrderID field of the OrderDetails table.

FK\_OrderDetails\_Products

This constraint allows only an existing product within the ProductID field of the OrderDetails table.

Relationship Script

The following script will add the constraint:

```
ALTER TABLE [OrderDetails] WITH CHECK ADD CONSTRAINT [FK_OrderDetails_Orders]
FOREIGN KEY([OrderID])
REFERENCES [Orders] ([OrderID])
```

Orders

The Orders table has only a single constraint or relationship. Examine Table 8-13; an explanation follows.

Table 8-13. Orders Relationships

Constraint Name	Foreign Key	Reference Table	Reference Field
FK_Orders_OrderStatus	OrderStatusID	OrderStatus	OrderStatusID

FK\_Orders\_OrderStatus

This constraint allows for a status type within the Orders table to be that of an existing status ID only.

Relationship Script

The following script will create the constraint:

```
ALTER TABLE [Orders] WITH CHECK ADD CONSTRAINT [FK_Orders_OrderStatus]
FOREIGN KEY([OrderStatusID])
REFERENCES [OrderStatus] ([OrderStatusID])
```

EndUser

The EndUser table has three separate constraints and relationships, as described in Table 8-14.

Table 8-14. EndUser Relationships

Constraint Name	Foreign Key	Reference Table	Reference Field
FK_EndUser_Address	AddressID	Address	AddressID
FK_EndUser_ContactInformation	ContactInformationID	ContactInformation	ContactInformationID
FK_EndUser_EndUserType	EndUserTypeID	EndUserType	EndUserTypeID



### FK\_EndUser\_Address

This relationship ensures that the AddressID field will be that of an existing value within the Address table.

### FK\_EndUser\_ContactInformation

This relationship ensures that the ContactInformationID field will be that of an existing value within the ContactInformation table.

### FK\_EndUser\_EndUserType

This relationship ensures that the EndUserID field will be that of an existing value within the EndUserType table.

## Relationship Script

The following script will create the constraints:

```
ALTER TABLE [EndUser] WITH CHECK ADD CONSTRAINT [FK_EndUser_Address]
FOREIGN KEY([AddressID])
REFERENCES [Address] ([AddressID])
```

GO

```
ALTER TABLE [EndUser] WITH CHECK ADD CONSTRAINT [FK_EndUser_ContactInformation]
FOREIGN KEY([ContactInformationID])
REFERENCES [ContactInformation] ([ContactInformationID])
```

GO

```
ALTER TABLE [EndUser] WITH CHECK ADD CONSTRAINT [FK_EndUser_EndUserType]
FOREIGN KEY([EndUserID])
REFERENCES [EndUserType] ([EndUserID])
```

## Products

The Products table has two individual relationships, as described in Table 8-15.

**Table 8-15.** *Products Relationships*

Constraint Name	Foreign Key	Reference Table	Reference Field
FK_Products_ProductCategory	ProductCategoryID	ProductCategory	ProductCategoryID
FK_Products_ProductImages	ProductImageID	ProductImages	ProductImageID

### FK\_Products\_ProductCategory

This relationship ensures that the ProductCategoryID field will be that of an existing value within the ProductCategory table.

### FK\_Products\_ProductImages

This relationship ensures that the ProductImageID field will be that of an existing value within the ProductImages table.

### Relationship Script

The following script will create the constraints:

```
ALTER TABLE [Products] WITH CHECK ADD CONSTRAINT [FK_Products_ProductCategory]
FOREIGN KEY([ProductCategoryID])
REFERENCES [ProductCategory] ([ProductCategoryID])
```

GO

```
ALTER TABLE [Products] WITH CHECK ADD CONSTRAINT [FK_Products_ProductImages]
FOREIGN KEY([ProductImageID])
REFERENCES [ProductImages] ([ProductImageID])
```

## ShoppingCart

The ShoppingCart table has a single relationship, as described in Table 8-16.

**Table 8-16.** *ShoppingCart Relationships*

Constraint Name	Foreign Key	Reference Table	Reference Field
FK_ShoppingCart_Products	ProductID	Products	ProductID

### FK\_ShoppingCart\_Products

This relationship ensures that the ProductID field will be that of an existing value within the Products table.

### Relationship Script

The following script will create the constraint:

```
ALTER TABLE [ShoppingCart] WITH CHECK ADD CONSTRAINT [FK_ShoppingCart_Products]
FOREIGN KEY([ProductID])
REFERENCES [Products] ([ProductID])
```

## Writing the Type Inserts

The *type inserts* for the database are a series of insert statements that will populate specific tables with values that will be required. For instance, you'll have a series of product categories along with different types with which a user will be associated.

The following sections show the insert statements that you'll need to execute against the database.

### EndUserType

The following script will create two records within the EndUserType table:

```
INSERT INTO EndUserType (TypeName) VALUES ('Customer')
INSERT INTO EndUserType (TypeName) VALUES ('Administrator')
```

### OrderStatus

The following script will create the records within the OrderStatus table:

```
INSERT INTO OrderStatus (OrderStatusName) VALUES ('Pending')
INSERT INTO OrderStatus (OrderStatusName) VALUES ('Shipped')
```

### ProductCategory

The following script will create the records within the ProductCategory table:

```
INSERT INTO ProductCategory (ProductCategoryName) VALUES('Appetizer Wine')
INSERT INTO ProductCategory (ProductCategoryName) VALUES('White Wine')
INSERT INTO ProductCategory (ProductCategoryName) VALUES('Red Wine')
INSERT INTO ProductCategory (ProductCategoryName) VALUES('Desert Wine')
INSERT INTO ProductCategory (ProductCategoryName) VALUES('Glasses')
INSERT INTO ProductCategory (ProductCategoryName) VALUES('Accessories')
INSERT INTO ProductCategory (ProductCategoryName) VALUES('Membership')
```

## Examining the Complete Database

You have completed the database setup and design for the LittleItalyVineyard database, its tables, and all the necessary relationships between those tables. At this point, it is helpful to look at the entire picture from a macro viewpoint. In other words, examine Figure 8-30, which shows all the tables along with the constraints.

---

**Note** As a result of the number of tables, you may not be able to see all of the details in Figure 8-30. However, within the book's downloadable code and accompanying files, you'll find a Portable Document Format (PDF) file of this database diagram.

---

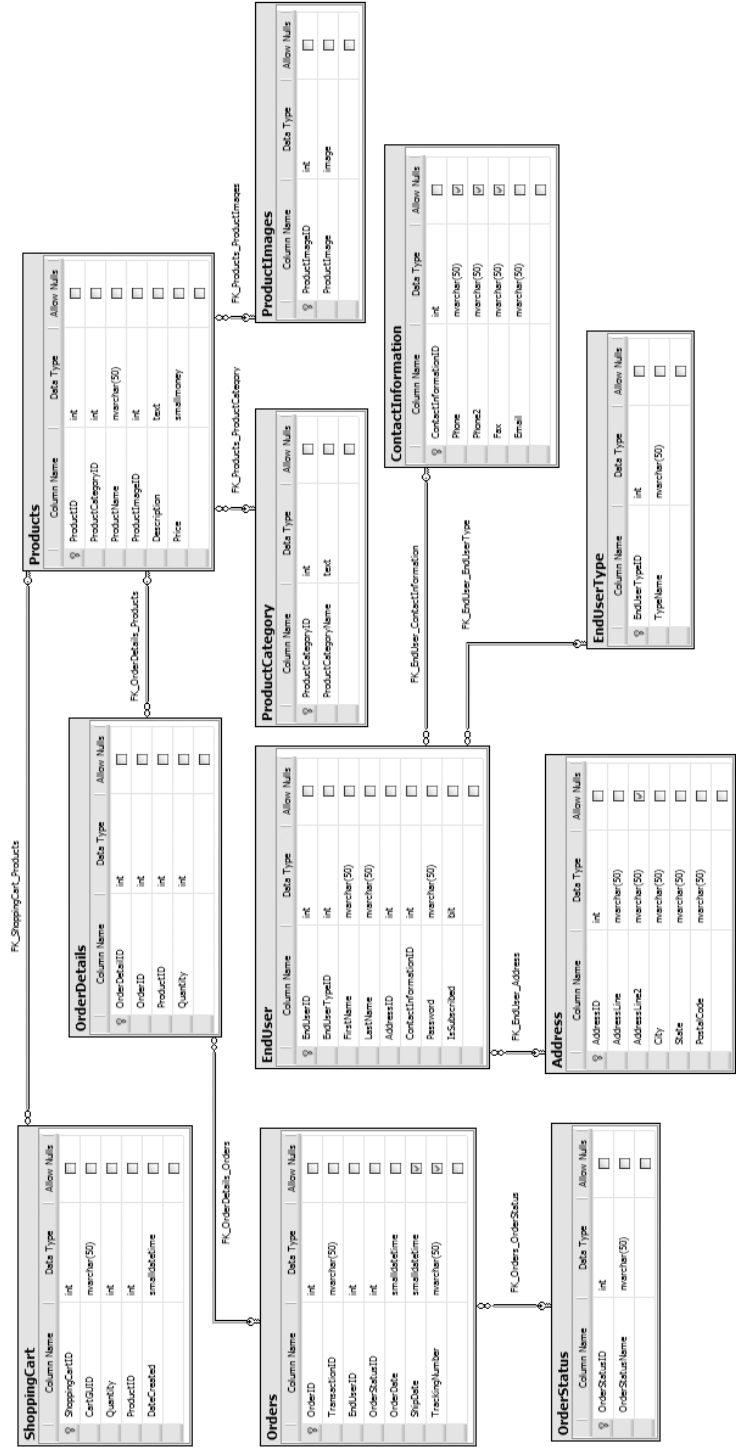


Figure 8-30. The database diagram

## Summary

This chapter showed you how to create the LittleItalyVineyard database along with the individual tables within the database. Then you established the relationships between certain tables.

One important aspect to remember is that although you have completed most of the work for the database, you might alter the database later in the development phase, based on details you flesh out during the development process. This practice is totally acceptable; however, you have established the main schema of the database, so any changes will be minor—more like tweaking than reconstructing the major facets of the database foundation.

