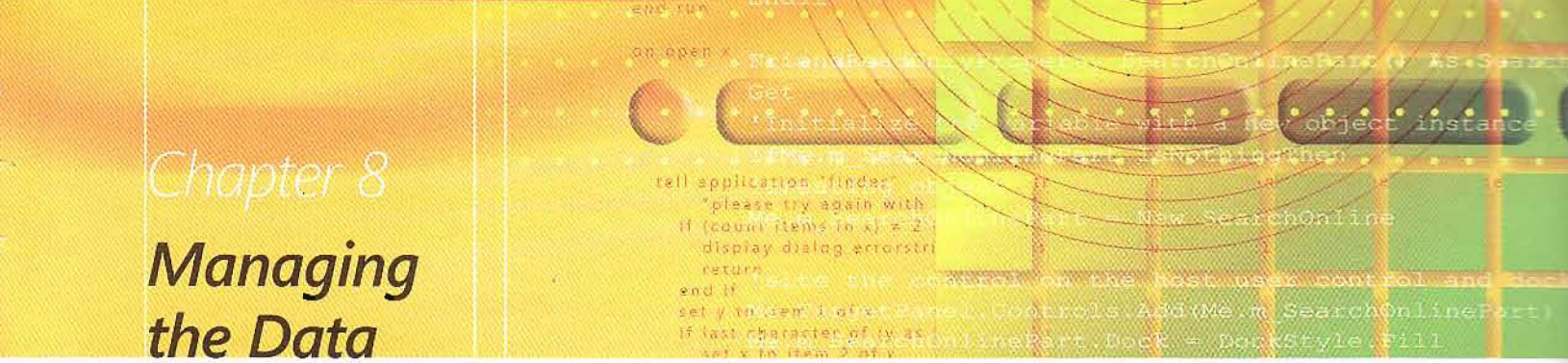


Chapter 8

Managing the Data

What Is a Database?, 128

SQL Server 2005 Express in Visual C# 2005 Express Edition, 136

What Are ADO.NET and Databinding?, 148

So far, you've seen how to build a Windows Forms application and what characteristics those applications contained, but you have not managed a great deal of data. Managing data is always a concern, whether at home, at the office, at school, or even for recreation. For instance, I have many recipes and ideas for great dinners, but when I want to prepare a nice meal, it takes me so much time to find them that usually I change my mind. If I had this information in my computer, it would be easy to quickly access my recipe for "rack of lamb with herb crust" and prepare a fabulous meal. I could also add other pertinent information to the recipe file, such as what side dishes were served with it or what wines went well with this recipe. I could even add a picture of the finished meal.

You could manage data using a word processing program, such as Microsoft Word, but it could become unmanageable as soon as you collect a lot of recipes and need to search for information within that file. Using a spreadsheet, such as Microsoft Excel, is also problematic. The fact is that trying to find information quickly when using more than one variable is close to impossible. Using the recipe example, suppose you want to retrieve all of the recipes that can serve at least six people and that have lamb stew meat but no mint in the ingredients because one of your guests is allergic to mint. Imagine the time it would take to find that information in either a Word file or an Excel spreadsheet. That's where databases come to the rescue.

In this chapter, you'll learn what a database is; how to create a database; how to add, delete, and update data; how to search or query a database; and how to use a database in a Windows Forms application. Accompanying Visual C# 2005 Express Edition is SQL Server 2005 Express Edition, which is a fully workable version of its bigger brother, SQL Server 2005, but with fewer features. SQL Server 2005 Express Edition is free, easy to use, and geared toward building simple and dynamic applications.

What Is a Database?

A database is a collection of data that is stored in files on disks using a systematic structure. The systematic structure enables users to query the data using management software called Database Management System (DBMS). SQL Server 2005 is a Relational Database Management System (RDBMS). It is based on a relational model because its data is structured using sets (the sets theory in mathematics) and logical relations (predicates). Most commercial database products are based on the relational model. In fact, it's been one of the most popular models for the last 20 years. Apart from Microsoft SQL Server, you may have also heard of the following product names: Oracle or IBM DB2.

What's In a Database?

NOTE

You'll learn about some of the other elements contained in a relational database later in this chapter.

A relational database, such as SQL Server 2005, contains multiple tables that are related together. A database can also contain views, stored procedures, functions, indexes, security information, and other elements. In this section, you'll learn about the basic element of a relational database, which is a table and its components.

A table contains columns and rows. A column defines the type of data, and a row contains the actual data. Because the relational model has strict rules, a RDBMS that uses the relation model must implement them.

MORE INFO

In reality, no popular RDBMS is fully implementing the pure relational model as it was first created in the 1970s.

Data Normalization and Data Integrity

The rules defining the relational model are called normalization rules. Normalization is a process that data architects must apply whenever they are at the design phase. Normalization rules exist to reduce the chance of having the same data stored in more than one table; in other words, they exist to reduce the level of redundancy and also to preserve data integrity in the database. Logiçally, the normalization process exists to help split the data into its own table so that there is no duplication of information in more than one table. For example, having an application in which the customer's address, city, state or province, zip or postal code, and country are duplicated in two different tables is a bad idea. There should be only one link from the customer table to the other table referencing additional customer information. Having duplicate data would make updates and deletions more problematic and would also pose the risk of having modified data in one table and not the other. This example demonstrates a data integrity problem.

Let's look at another data integrity problem. Suppose you have both a product table and a table containing customer order details. Although you normalized your data, data integrity does not exist (for this example). Now let's say you decide to delete product1, which means removing a row from the product table that corresponds to product1. If the RDBMS would let you do this, it would mean that suddenly all rows in the customer order details table that contained this product would not be able to show which product was ordered because the product would no longer exist. Those rows would be orphaned, which could have disastrous results for the company.

As you can see, data integrity is a very important concept that is related to the accuracy, validity, and correctness of the data. To better understand some of these concepts, let's look at another example.

Suppose you are the owner of an online store and want to manage your company using a software application. To use a software application, you must start thinking about using a database. Any company, both small or large, typically has a great deal of data to store. Also, because data is all around us, people want more access to this data so as to create reports and conduct analysis. That is why databases are so useful. Returning to your online store, at a minimum you would like to store information about your customers, products, invoices, purchasing, and inventory. To summarize all of those areas, let's take a look at the Product, OrderHeader, and OrderDetail tables.

NOTE

The following tables have purposely been kept simple (some columns are missing) and are used to illustrate the concepts you've just learned.

Column Name	Data Type	Allow Nulls?
ProductID (PK)	integer	Not Null
ProductNumber	nvarchar(10)	Not Null
Name	nvarchar(50)	Not Null
Description	nvarchar(200)	Null
Photo	image	Null
Price	money	Not Null
Taxable	bit	Not Null

Table 8-1
Product Table

Column Name	Data Type	Allow Nulls?
OrderID (PK)	integer	Not Null
OrderDate	datetime	Not Null
DueDate	datetime	Not Null
CustomerID (FK)	integer	Not Null
TaxAmount	money	Not Null
Total	money	Not Null

Table 8-2
OrderHeader Table

Column Name	Data Type	Allow Nulls?
OrderID (PK) (FK)	integer	Not Null
LineDetailID (PK)	integer	Not Null
ProductID (FK)	integer	Not Null
Quantity	integer	Not Null
LineTotal	numeric(38,6)	Not Null

Table 8-3
OrderDetail Table

Your Product, OrderHeader, and OrderDetail tables could also be represented graphically, as shown in Figure 8-1. This is a common way of looking at databases.

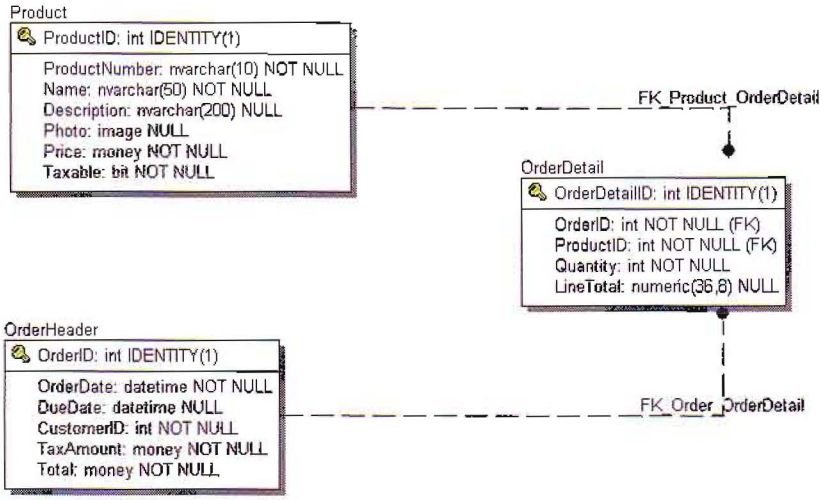


Figure 8-1
Partial database diagram of a small online company

What Is Null?

One of your first observations is that there's a column in the tables titled Allow Nulls?, which is also reflected in Figure 8-1. When designing a table, you need to consider what's absolutely necessary (Not Null) and what's not (Null). For instance, when you insert a new row into the Product table, it may not matter whether the product has a photo, but it might be a problem to have a product without a product number. Now let's correlate how allowing null is related to data integrity. Whenever a table is designed with a column that doesn't allow null, the RDBMS will reject any insertion of a new row that has a column set to null when it is not supposed to. When you pay attention to those columns that cannot contain null when designing your tables, you automatically add another data integrity layer by making sure that all necessary data is present before the record is inserted into the database.

What Are Primary Keys and Foreign Keys?

You'll see in Figure 8-1 that some columns have the letters (PK) for primary key in the tables or a yellow key for primary keys. Some columns also contain (FK) for foreign key. Let's start by talking about the primary key.

Primary Key

A primary key is a value that is used to uniquely identify a specific row in a table. A primary key:

- Can be composed of one or more column names. When it's composed of more than one column, it's called a composite key.
- Is often a numeric field.
- Is often generated by the RDBMS, in which case it's called a surrogate key. A surrogate key is frequently (but not always) a sequential number. A surrogate key is also called an identity in SQL Server 2005. An identity starts at a set number, called the identity seed, and increments by another set number, called the identity increment. For example, if you create a table named Product, you can have a column named ProductID that is set as an identity, and you can set the identity seed to 1 with an identity increment to 1. When the first row is created in the Product table, the ProductID will be generated by the RDBMS and set to 1. The following row will have a ProductID that is set to 2 and so forth.

- Should be as short as possible, but long enough to support the number of rows it will represent.
- Is immutable, meaning its value should never change.
- Is also a natural key when the key has a logical relationship with the rest of the columns in the table. For example, if you had a book table, the ISBN number could be used as a primary key because it uniquely identifies only one book. It would be an advantage compared with a generated key because it would take less space and has to exist anyway!
- Is also used to relate two tables together.

In our Product table example, the ProductID is the primary key. At design time, it will also be an identity. You can claim that the product number could be a primary key and you could be right, but in certain scenarios a product number could be used twice. For example, suppose you have product #FG-001 with a revision 1.0. In time, you change the product because of customer complaints and give it a revision 2.0. You want your customers to continue to order the same product number for many business reasons. In your database, you would retire the product revision 1.0 by perhaps changing a column named Active, then add another row in your table with the new product details, including revision 2.0, and set it to Active. Why can't you use the same row? Let's assume that six months after creating the new product revision, you want to create a graph to determine whether your changes to the product meant that you had fewer returns from your customers. It would be difficult to come up with good data if you had only one row for the product, but it would be fairly easy to do if you have two rows because they would be unique in the database, with each one having a different ProductID.

In the OrderDetail table, you have a composite primary key that is a combination of the OrderID and OrderDetailID. This means that these two columns would ensure the uniqueness of a row in the OrderDetail table.

In the OrderHeader table, the OrderID is the primary key.

Foreign Key

A foreign key is a column in a table that relates to a column in another table. It also enables you to create relations between tables. A foreign key in a table is always a primary key in another table. Foreign keys are used to enforce data integrity by being part of foreign key constraints. Foreign key constraints are created to make sure referential integrity is preserved and not violated. There are two foreign keys in the order details. The first is the ProductID foreign key in the OrderDetail table, and it's related to the primary key named ProductID in the Product table. The second is the OrderID foreign key in the OrderDetail table, and it's related to the primary key named OrderID in the OrderHeader table. Concerning the naming of foreign keys, it's a good practice to define them using the same name as their primary key counterpart; otherwise, it may lead to problems for those looking at your logical data model.

I introduced you to data integrity at the beginning of this chapter. In doing so, I cited an example that could create similar problems to the one in the Product and OrderDetail table example. Adding a foreign key constraint between these two tables would prevent a user from deleting a product in the Product table that could potentially create a large number of orphaned rows in the OrderDetail table. If you look at Figure 8-1, the foreign key constraint between Product and OrderDetail is shown as a line between the two tables that can be found by looking at the name FK_Product_OrderDetail. Naming constraints is an easy way to understand what they are for. We only have three tables in our example, but you can imagine that constraints without names that exist between numerous tables would quickly become unclear.

Another foreign key constraint exists here, which is the one between the OrderHeader and OrderDetail tables that would prevent an order from being deleted before all of its matching OrderDetails have been deleted. You can see in Figure 8-1 that the OrderHeader table has another foreign key called CustomerID. Therefore, another foreign key constraint would exist between the Customer and OrderHeader tables. Following the same principles found with other foreign key constraints, this would prevent a customer from the Customer table from being deleted before all of its matching orders in the OrderHeader table and all detail rows in the OrderDetails table that match the orders have been deleted.

If there were no foreign key constraints in this database, data integrity would be easily violated. The database would be left with a big problem: a time bomb of orphaned rows

that take up space and slow down all queries. By adding this foreign key constraint, the RDBMS would ensure that all rows in the OrderDetail table that reference this product have been deleted before the product row could be deleted in the Product table.

How Do You Interact with a Relational Database?

So far, I've talked about tables in which you can update, add, or delete rows or query the database to get particular results. Perhaps you've been asking yourself: But how do I talk or interact with the database? How does it return the answers to my queries? And how do you create those tables? I'm sure you've been asking yourself many other questions as well. The answer to all of these questions is SQL Server 2005 Express Edition.

SQL stands for **S**tructured **Q**uery Language and was invented in the 1970s. The acronym is pronounced SEQUEL and was also introduced using that same spelling, but because of a trademark dispute in the UK in the 1970s, the name was shortened to the now well-known SQL acronym. Back then, the SEQUEL acronym meant **S**tructured **E**nglish **Qu**ery **L**anguage. SQL is an English-based language and is very similar to human language questions. That's why it's easy and fast to learn basic SQL programming. Let's look at two examples:

1. SELECT * FROM CUSTOMER
2. SELECT COUNT(*) FROM PRODUCT

The first example can be translated in English to give me all (*) rows in the Customer table or give me the list of customers in English. The second example can be translated as a request to give me the total of all rows contained in the Product table or to count how many products this company has.

When you issue an SQL query to a relational database, the database returns a result set that simply contains the rows with the answers to your query. Using SQL, you can also group or aggregate the results of a query. You also use SQL to create tables or delete (drop) tables. You've learned about primary keys, foreign keys, and constraints, but you probably didn't know that they're also created using SQL.

It's also good to know that SQL is an ANSI/ISO standard; therefore, any RDBMS producer needs to obey a set of rules. Basic SQL is a base programming language and as such is usually not sufficient to solve all possible problems or analysis needs that an application may demand. It has a rather limited set of keywords. Because its first goal is to query data from a

database, the most popular RDBMSs on the market have added extensions to SQL to permit the addition of procedural code. These additions turn SQL into a full-fledged programming language that helps solve more complex problems. The following is a list of popular extensions and their manufacturers: Microsoft Transact-SQL (or T-SQL for short), Oracle PL/SQL, and IBM SQL PL. Recently, in addition to these extensions, RDBMS manufacturers have added the support of other programming languages. Microsoft is adding .NET language support into the database with all SQL Server 2005 Editions, while Oracle and IBM have added Java support.

There are more database concepts and theories than those listed and explained here, but we have covered the immediate database needs of this book. You'll now apply those concepts concretely in a Windows Forms application that will use a SQL Server Express 2005 database.

SQL Server 2005 Express in Visual C# 2005 Express Edition

In this section, you will develop a Windows Forms application. This will be a car tracker application that will enable the user to track the prices of cars over time and determine where the listing was observed. You will first use Visual Studio to create the database and the tables, then add some data and validate some of the concepts you've learned in the first part of this chapter. You will then create a Windows application that will use your data and build a data-centric application that will allow the user to store any amount of data.

Refer to Figure 8-2 for the database diagram pertaining to this section's example.

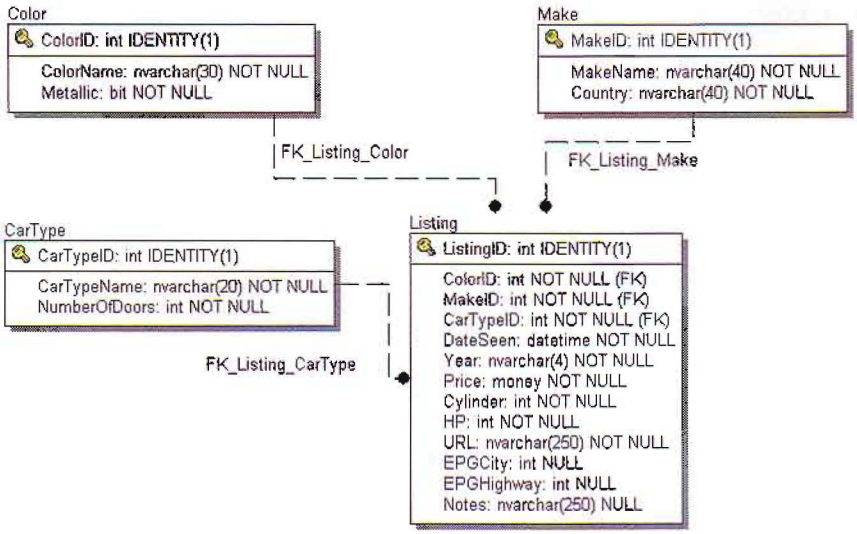


Figure 8-2
Car tracker application database diagram

Creating a Database Using Visual C# 2005 Express Edition

Before using data, we need a place to store the data. You'll learn how to create a database in Visual C# 2005 Express Edition. You'll also see how easy it is for you to create all of the tables we need to satisfy the needs of our car tracker application because the SQL Server team did a wonderful job of integrating the tools into Visual Studio.

TO CREATE A DATABASE USING VISUAL C# 2005 EXPRESS EDITION

1. Start **Visual C# 2005 Express Edition**.
2. Create a new Windows application and name it **CarTracker**
3. You will now create the database that will hold all the tables for the application. In the Solution Explorer, right-click the **CarTracker** project, select **Add**, and then select **New Item**
4. In the Add New Item dialog box, select **SQL Database** from the Visual Studio Installed Templates. Type in the filename **CarTracker.mdf** and click the **Add** button. By doing so, you'll create a database and attach the database file (**CarTracker.mdf**) to your CarTracker project.
5. You will then see a Data Source Configuration Wizard. Don't pay attention to this dialog box just yet, you'll learn about it soon. Just click **Cancel** for now.

MORE INFO

SQL Server is well integrated because Visual Studio provides a great SDK for other components to plug into the IDE.

MORE INFO

The .mdf file extension is used by the SQL Server family of products. The .mdf contains the entire database; this means all tables and other elements that can exist in the database are located in that one file. The only thing that is not part of the .mdf file is the log information, which is in a .ldf file. It is created whenever you create the database. The file has an .ldf extension and is used to store the database log information. You can see this file by clicking the Show All Files icon in the Solution Explorer.

6. The Solution Explorer should now contain a new item within your project: the new database file called **CarTracker.mdf** as shown in Figure 8-3.

Figure 8-3
Solution Explorer with the newly created CarTracker.mdf database file

7. You will now start to add tables to your database. To do this, you can either double-click the **CarTracker.mdf** file or right-click **CarTracker.mdf** and then select **Open**. This will cause Visual Studio to connect to the SQL Server 2005 Express instance installed on your machine.

Figure 8-4
Database Explorer with the CarTracker database connected

The Database Explorer should appear on the left side of the screen where the toolbox usually opens up, as shown in Figure 8-4. If you do not see the Database Explorer, go to the View menu, Other Windows, Database Explorer.

Under the database name, you should see a list of database elements represented by folder icons. Although you will not recognize most of them, you will see two elements that are already familiar to you: the database diagram and the tables. You will use both of these elements shortly.

You'll know that you're connected to the database when you see the database icon with an electric cord look-alike. When you're disconnected, you will see the database icon with a red X. However, seeing a red X does not necessarily mean that you're disconnected. You might have been disconnected earlier, but the Database Explorer was never refreshed. To verify the state of the connection to your database, you should click the **Refresh** button in the Database Explorer toolbar.

8. Right-click on your database named **CarTracker.mdf** in the Database Explorer and select **Close Connection**. You should now see the red X near your database name.

You're now disconnected. To reconnect, you can do three different things. You can double-click your database name (e.g., **CarTracker.mdf**) in the Database Explorer, you can click the Refresh button, or you can right-click on the filename in the Database Explorer and select **Modify Connection. . .**. If you select the Modify Connection route, you will see a dialog box like the one shown in Figure 8-5.

Figure 8-5
The Modify Connection dialog box will let you reconnect to your CarTracker database

9. Because it's a good practice to test your connection, you can click the Test Connection button and it will verify the connection currently specified. It's also verifying that SQL Server 2005 Express Edition is ready and able to receive connections from your applications.

NOTE

Currently, you have only one database in your projects, but it's not unusual to need to connect to and get information from two or more databases. That's why Database Connections in the Database Explorer is there as a tree, for it's representing each database as a node in that tree. You have only one node in the tree, which is your CarTracker database.

10. Click **OK** to reconnect to your database.

One thing you should pay attention to in this dialog box is the Database Filename. Because you didn't save any files, everything is still located in a temporary folder identified by the content of the text box. As soon as you save all of the files in your project, the database will be saved along with the other project files, wherever they are located. You can later verify that location by going to the Tools menu, selecting Options. . . , and then looking at the Projects and Solutions node in the tree. On the right panel in this dialog box, you can determine where your projects are stored by looking at the first text box called Visual Studio Project Locations.

Creating Tables in Your Database

Now you'll create all tables and relationships needed for the CarTracker application. Using the information found in Figure 8-2, you'll create tables, primary keys, identities, and foreign key relationships in the CarTracker database, and you'll do all of this without leaving Visual Studio.

TO CREATE TABLES IN A DATABASE

1. Let's start with the Color table. In the Database Explorer, right-click on the table's folder icon and select **Add New Table**. You should now see an empty grid on the designer surface, which is called the Table Designer. You will also see that a new toolbar has appeared, which is called the Table Designer toolbar. This toolbar has all the tools necessary to help you create a table without writing a single SQL query.

2. You'll now add a column to the Color table. Type **ColorID** in the Column Name field of the Table Designer. Select **int** as the Data Type and uncheck the **Allow Nulls** check box because this column will be the primary key in this table. A primary key cannot be null since it is part of the uniqueness of a row in the table.

3. Before you add the second column in the Color table, you'll set this column as the primary key. To do so, you need to click the **Set Primary Key** icon in the Table Designer toolbar.

The database diagram shown in Figure 8-2 illustrates that you also need this column to be an identity; therefore, you need to modify that property in the Column Properties window right below the Table Designer. Scroll down until you see the Identity Specification group. Click on the **plus sign** (+) located to the left of the words *Identity Specification* to expand this group. Now click in the **(Is Identity)** field and set it to **Yes**. Leave both the Identity Seed and Identity Increment set at 1 for now

NOTE

From this point onward, for every tree control and every control that is a group (i.e., has a + sign), the word *expand* will be used instead of repeating the words *click on the + sign*.

NOTE

As a reminder, when a column is an identity, SQL Server automatically generates a new number each time a row is created in a table. It starts at the value indicated by the Identity Seed property and increases in increments by the value indicated by the Identity Increment property.

4 To add another column, click in the row under the ColorID column name. Add the two remaining columns based on the diagram shown in Figure 8-2. You can set the size of the ColorName nvarchar to 30, by typing in the Data Type field. When done, your table should look like the one shown in Figure 8-6.

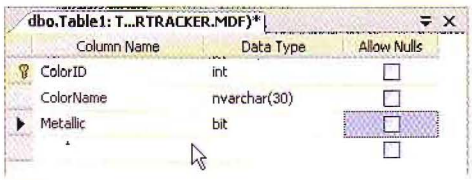


Figure 8-6
Table Designer with all of the columns for the Color table

5 Now that you're done with the design, you need to add the table to the database. To do this, you need to save the table. Click the **Save** icon or press **Ctrl+S**. When the Choose Name dialog box appears, as shown in Figure 8-7, name your table **Color** and then click **OK**.

NOTE

In the Table Designer, the little black triangle indicates the current row.

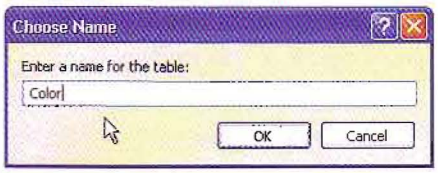


Figure 8-7
The Choose Name dialog box showing the Color table name

Figure 8-8
Database Explorer with the Tables folder and Color table expanded

6 Expand the **Tables** folder in the Database Explorer to view the list of existing tables in the database; the new Color table should appear. When you expand the **Color** table to view the list of columns, all three columns that you just created should appear, as shown in Figure 8-8.

7 Close the Color table in the Table Designer by clicking the **X** near the Solution Explorer.

8 Click the **Save All** icon in the toolbar to save your project. Make sure the project name is CarTracker and click the **Save** button.

9 Before creating other tables, read this step completely. Now that you have the knowledge to create a table, create all remaining tables (ColorType, Make, and Listing) using the same techniques you've just learned. Make sure that all tables and ***all*** of their columns are recreated exactly the same way in your tables as shown in Figure 8-2. Don't worry about establishing the relationships, for you'll create those in the following exercises. Between each table creation, save your new table immediately and make sure it appears in the Database Explorer. Then close the table in the designer surface as shown earlier in step 7 of this section.

TIP

Whenever you click on a column name in the Database Explorer, you'll see the properties listed in the Properties window. This is the same Properties window that you've been using with one minor difference: it is a read-only view and therefore does not let you modify information.

Creating Relationships Between the Tables

You have created tables, but they don't have any relationships. You'll now add those relationships and make sure your database has data integrity to cover the basis of orphaned rows. Like many other elements in Visual C# 2005 Express Edition, there's more than one way to create those relationships. One is more visual than the other, and you'll start with this more visual approach so as to stay focused on the main idea of the book, which is being productive.

Before you're able to create the relationships visually, there is a prerequisite to add to your project: a database diagram. It might not look exactly as the one shown in Figure 8-2, but it will be similar.

TO CREATE RELATIONSHIPS BETWEEN TABLES

1. Go to the Database Explorer and right-click the **Database Diagrams** node located above the Tables node. Select **Add New Diagram**. A dialog box will appear indicating that SQL Server 2005 Express Edition doesn't have all of the database objects it needs if you want to create database diagrams.

2. Click **Yes** to have SQL Server create the components it needs to obtain a database diagram. When it's done creating, you should be asked which tables you want to add to your diagram in the Add Table dialog box.

3. Select all of the tables you created and then click **Add**. It should take less than a minute for your diagram to appear. Click the **Close** button to indicate to Visual Studio that you have all the tables you need

4. Click the **Save All** button or press **Ctrl+Shift+S**. You'll be asked to save your diagram and choose a name. Name your diagram **CarTrackerDiagram**.

5. If you don't see your database diagram, first go to your **Database Diagrams** node, expand it, and then open the diagram by double-clicking on it. You should see the designer surface with all of your tables.

TIP

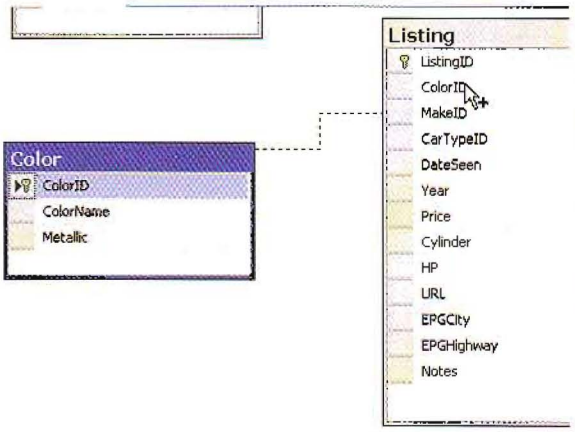
Depending on your resolution, the view might be tight. If you want to view more of the diagram, you might need to unpin or close some windows, such as the Solution Explorer or the Properties window; you can return these items to your screen by going to the View menu and selecting Solution Explorer or the Properties window. You can also change the zoom value by changing the value in the Zoom drop-down list.

Let's focus on one relationship that we need to create. When you look at Figure 8-2, you'll see that the ColorID column is present in the Listing table because there's a relation to the Color table. The line between both tables is a foreign key (FK) relationship. You need to have this relationship established or otherwise you'll have orphaned nodes in the Listing table whenever a Color row is deleted. This means that you have to establish a relationship between the primary key table and the foreign key table. In this case, it means you need to create a relationship from the Color table toward the Listing table.

6 In the database diagram, click on **ColorID** in the Color table where you see the small yellow key.

7 Look at Figure 8-9 to see where you should be at the end of this manipulation. Hold the left button down and drag **ColorID** toward the Listing table, you should see a line appear as you drag. Align your mouse cursor so that it's over the column with which you want to create the relationship—in your case, over the ColorID field in the Listing table. When you see a small + appear, then drop it.

Figure 8-9
Creating the foreign key relationship between the CarType and Car tables



8 If you correctly selected and released the mouse once you were over ColorID in the Color table, you should see a Tables and Columns dialog box that asks you to confirm the creation of the FK relationship. It's important for each table that ColorID is the column name that appears to link both tables in that dialog box. If the primary key and foreign key tables are correct and the selected column names are correct, then click the **OK** button.

9 You should then see the Foreign Key Relationship dialog box shown in Figure 8-10.

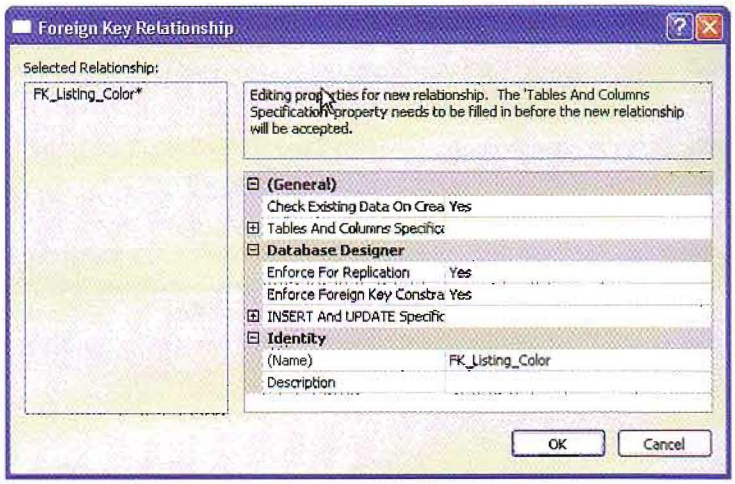


Figure 8-10
Foreign Key Relationship dialog box for the Listing to Color tables

10 Although you can change some properties within this dialog box, just click **OK** for now. See Figure 8-11 to view the diagram with the new relationship created.

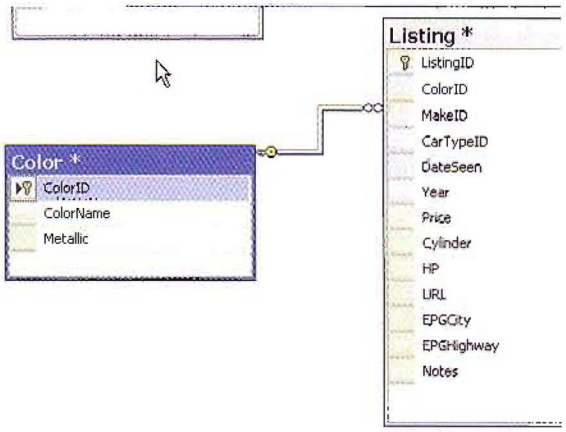


Figure 8-11
Modified diagram showing the new FK relationship between the Listing and Color tables

The first thing to note on the diagram is the infinity symbol located close to the Listing table and the yellow key located close to the Color table. The infinity symbol on the Listing table indicates the table's cardinality. It indicates that, in this relationship, the Listing table can contain many rows with information coming from the matching primary key table. The yellow key indicates from which table the primary key is coming.

I rearranged the diagram so that the two tables are close together. You can rearrange your tables any way you want by dragging them by the title bar (i.e., where the table name is displayed). This is sometimes necessary to do when you create relationships so that you do not end up with an unusual-looking diagram. I suggest that you put your Listing table in the middle of your other tables because it will be easier to create relationships this way. You can also rearrange the tables on your diagram at any time by right-clicking anywhere except on a table on the diagram's designer surface and selecting **Arrange Tables**. You can also have the labels for every relationship appear on the diagram by right-clicking the diagram's designer surface and selecting **Show Relationship Labels**, as shown in Figure 8-12.

MORE INFO

To reinforce the concept of establishing relationships between tables, let me give you another way of looking at the relationship in this exercise. There are two reasons why the ColorID column is in the Listing table as an FK. The first reason is that it is used for a normalization and design principle because you don't want to have duplicate data. The second reason is that it is used for data integrity reasons and, more specifically, for the orphaned rows problem. Let's look at it with some sample data. Suppose there is a Color row called Dark Blue and the Listing table contains six different ad definitions that are Dark Blue. If you remove the Dark Blue color from the Color table, it would mean that those six ads would have orphaned data. That is why you created a foreign key relationship: to make sure that if an application or a user tries to remove data in the Color table, a process within SQL Server 2005 would prevent this by validating that there are no "kids" left behind in the Listing table before allowing the deletion to occur in the Color table.

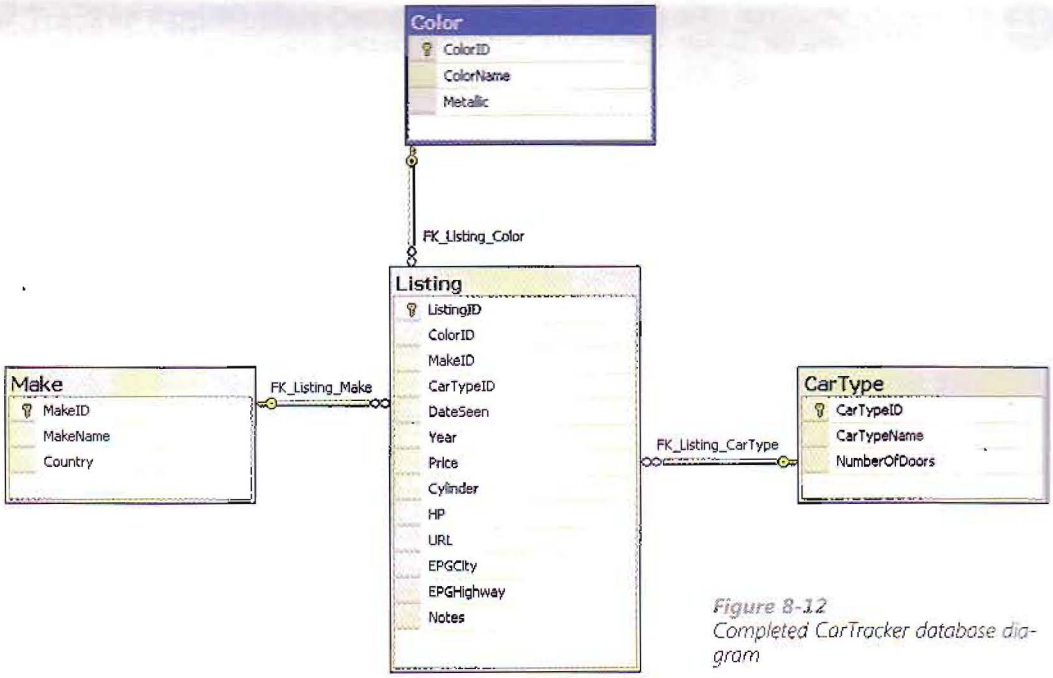


Figure 8-12
Completed CarTracker database diagram

11 Now create the other FK relationships by using either Figure 8-2 or the following table.

Table 8-4
List of foreign key relationships to create

Column	Primary Key Table	Foreign Key Table
MakeID	Make	Listing
CarTypeID	CarType	Listing

MORE INFO

You can always go back and review the properties of any relationship by double-clicking on the line or by right-clicking and selecting Properties **from the context-sensitive menu.**

When finished, the content of your diagram should resemble the content shown in Figure 8-12. Make sure your relationships are arranged properly by looking at where the infinity symbols and yellow keys are located and also by looking at the previous table for verification.

12 Click the **Save All** button or press **Ctrl+Shift+S** to commit the changes to the database. Click **Yes** when asked if you want to save.

Entering Data in SQL Server Tables Using Visual Studio

Now that you have created all of your tables and relationships, you'll start inserting data in your tables and verifying that your constraints ensure the data integrity of your database.

Let's start by adding data in all tables. You'll first add rows into the Color table.

TO ENTER DATA IN SQL SERVER TABLES USING VISUAL STUDIO

1. To start entering rows in the Color table, right-click the **Color** table in the Database Explorer and select **Show Table Data**. Your designer surface should have a grid like the one shown in Figure 8-13.

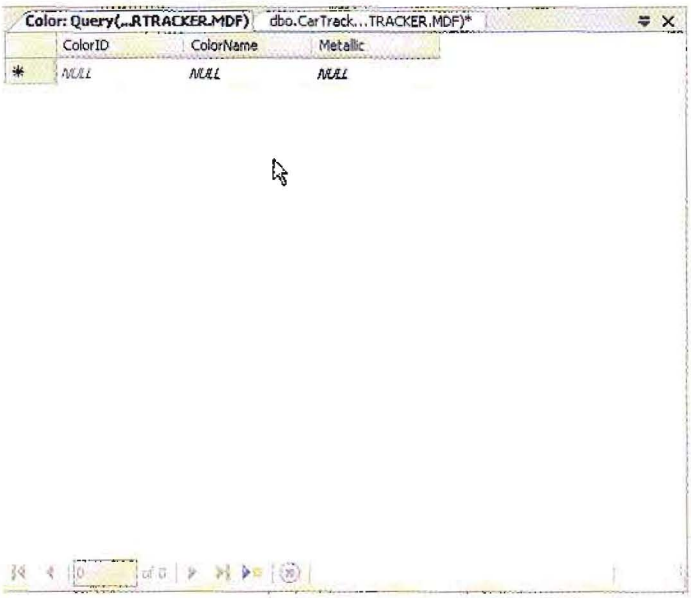


Figure 8-13
Empty Color table in the table data grid

2. Let's add the first color. Click in the **Color Name** field, type **Dark Blue**, and then press the **Tab** key to go to the next column. Type **true** in the **Metallic** field. Because that column type is a bit, its values can only be either true or false because a bit type is a binary type. When you're done, press the **Tab** key to go to the next row.

While typing your data, look to the table's left-most area in the table data grid a you'll see a small black trian , a pencil icon both a star and a small blac triangle or a star . The pencil indicates that you're making a modifi tion to the row. The star indi cates a new row and the sm black triangle indicates the c rent row.

Color: Query(...RTRACKER.MDF) | dbo.Color: Ta...RTRACKER.I

ColorID	ColorName	Metallic
1	Dark Blue	True
2	Red	True
3	Silver	False
4	Black	False
NULL	NULL	NULL

Figure 8-14
Color table with four new rows of data

3 Add three more car colors—**Red**, **Silver**, and **Black**—and set Red as **Metallic** and the other two colors as **Non-metallic** (i.e., false). When done, the table should look like the one shown in Figure 8-14

4 Add the following data to the Make and the CarType tables

Make Table

MakeName	Country
GoodRoadster	Germany
SmallCar	France
BigSUV	USA
ReliableCar	Japan

CarType Table

CarTypeName	NumberOfDoors
Roadster	2
SUV	5
Hatchback	5
Sedan	4
Coupe	2

For all columns that you created as an identity, don't type the data because the field will automatically be generated by SQL Server 2005 Express Edition whenever the row is created in the table. If you try to type data in an Identity column, you will not be allowed to do so. When in an Identity column, it states that the cell is read-only near the navigation bar at the bottom of the Table Designer.

You might not have realized that by giving a type to your data, you actually added data integrity verification to your database. Try modifying one of the Color rows by changing the Metallic column to read *Helloworld* instead of true or false. You'll get an error message telling you that the Metallic field is of type Boolean.

To show how data integrity is preserved using the foreign key constraints, you'll add two Listing rows. You will enter more rows when using your Windows Form application.

5 Right-click on the **Listing** table, select **Show Table Data**, and add the following two rows

Listing Table

ColorID	MakeID	CarTypeID	DateSeen	Year	Price	Cylinder	HP	URL	EPGCity	EPGHighway	Notes
1	1	1	08/11/2005	2005	42500	6	240	http://www.litwareinc.com/	20	28	This is my dream car, follow regularly.
4	3	2	07/30/2005	2003	39775	8	340	http://www.cpandl.com/	10	15	Too much gas

6. You'll now verify that one of your foreign key constraints is working correctly. Open the Make table by right-clicking on the **Make** table and selecting **Show Table Data**

7. Let's try to delete the first row by clicking on the left-most field where the pencil usually appears. The row should be selected and all fields should be blue. Right-click and select **Delete**.

8. A dialog box should appear on your screen inquiring whether you really want to delete the row. Click **Yes**.

9. You should receive a dialog box error message stating that the row was not deleted because of the foreign key constraint that reads as follows: **Error Message: The DELETE statement conflicted with the REFERENCE constraint "FK_Listing_Make."** This statement affirms why the foreign key constraint was created, which was to avoid orphaned rows. Figure 8-15 depicts what the error dialog box looks like and what kind of information is provided to help you debug the problem, if necessary. In this case, it's not a problem but a feature of your creation!

10. Click **OK** to exit this dialog box.

11. Test your other constraints related to the Listing table by trying to delete the first row of the CarType table. You should receive a similar error message.

MORE INFO

You can navigate through the table by using the navigation controls at the bottom of the grid. These controls will allow you to do things such as move to the first and last row, move to the previous and next entry, move to a new record, or directly type in the row number.

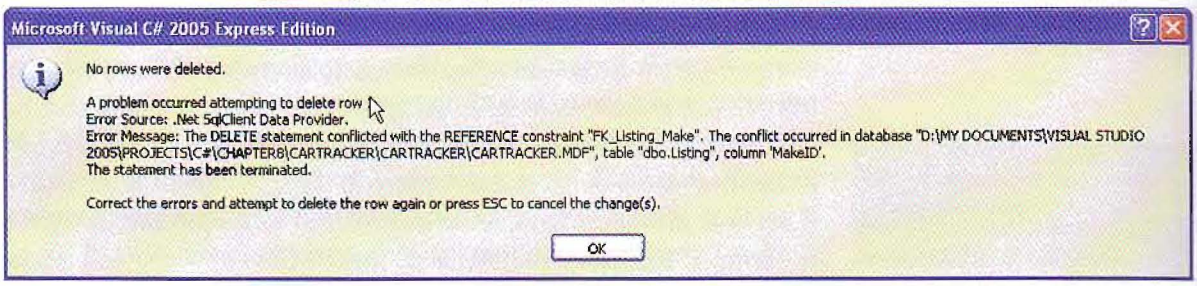


Figure 8-15
Error dialog box showing the foreign key relationship preventing the deletion of a row from the Make table

Now that you have all of your domain tables loaded with some data, you'll now learn to use the database in a Windows Forms application. You'll learn about ADO.NET and about databindings with Windows Form controls.

What Are ADO.NET and Databinding?

If you want more information about SQL and Transact-SQL, you should download the SQL Server 2005 Express documentation. You can find this information at the following link: http://go.microsoft.com/fwlink/?LinkId=51842. SQL Server 2005 Express documentation is designed to help you answer most questions you might have, but it might also refer you to the SQL Server 2005 documentation. You can download it at http://go.microsoft.com/fwlink/?LinkId=51843.

You rarely enter all data manually using Visual Studio. You typically let the user do it or you do it through an application. You can also either import data from another source or create the new data using SQL scripts, but this is a more advanced concept that will not be covered in this book.

This section will focus on how to build Windows applications that can connect to and receive data from a SQL Server 2005 Express Edition using ADO.NET. The following is a formal, official definition of ADO.NET from the MSDN online library:

*ADO.NET provides consistent access to data sources, such as Microsoft SQL Server, as well as data sources exposed through **OLE DB** and **XML**. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data.*

ADO.NET cleanly factors data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly or placed in an ADO.NET DataSet object in order to be exposed to the user in an ad hoc manner, combined with data from multiple sources, or remoted between tiers. The ADO.NET DataSet object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.

The ADO.NET classes are found in System.Data.dll and are integrated with the XML classes found in System.Xml.dll. When compiling code that uses the System.Data namespace, reference both System.Data.dll and System.Xml.dll.

I wanted to present the long and formal definition of ADO.NET because it contains elements that you'll learn about while working with the Car Tracker application. I also chose it because I would like you to refer back to it whenever you're working with ADO.NET. Here is a less formal definition that I think summarizes what ADO.NET is all about.

You can say that ADO.NET is the .NET Framework way of accessing and programmatically manipulating databases. With ADO.NET you can also manipulate other sources of data like XML sources.

New to ADO.NET 2.0 are new ways of accessing data from different sources. In Visual C# 2005 Express Edition, you are limited to the following data sources: databases (SQL Server Express and Microsoft Access databases), Web services, and custom objects. It is much easier (i.e., there is less code) to manipulate data in ADO.NET 2.0, especially when using all of the tools included in Visual Studio 2005. There are many new wizards and other tools that make the experience of working with databases a pleasant one. Visual Studio 2005 covers numerous common scenarios with its tools and wizards, but it's also very powerful when used programmatically without the use of the visual tools. You will learn the basics in this book, but there's nothing preventing you from learning more about databinding and ADO.NET and from unleashing powerful applications.

Visual C# Express allows you to work with Microsoft Access databases, but workin with SQL Server 2005 Expres Edition gives you all the ben- efits of the Enterprise quality SQL Server 2005, with the or downside being a reduced se features.

Before proceeding any further, let's talk about the Car Tracker application. The main goal of the application is to track car ads over the Internet. As you have your database ready to go, you now need to consider what will be included in this application. In reality, what you need is simply a way of displaying the ads, adding new ads, modifying/deleting existing ads, and searching through the ads using a series of drop-down boxes that allow you to narrow your search based on certain criteria. These search criteria will come directly from the domain tables (i.e., separate drop-down controls for the car type, color, make, and so forth).

When using drop-down controls or any other controls with data that you know exist in your database, you don't want to populate the data by hand. You want to use the databinding capabilities of a control. **Databinding** is an easy and transparent way to read/write data and a link between a control on a Windows Form and a data source from your application.

ADO.NET takes care of a great deal of activity behind the scenes (it's even better in .NET Framework 2.0), as well as managing the connection to the database. Managing the connection doesn't stop at opening and closing the connection, but also concerns itself with finding the database with which you're trying to connect. When a connection is opened, it means your application can talk to the database through ADO.NET method calls. All exchanges (send/receive) of data between your application and the database are managed for you by ADO.NET. The data itself is also managed by ADO.NET through diverse mechanisms: read-only forward navigation, navigation in any direction with read-write, field evaluation, and so forth. And the beauty of it is that you usually don't have to write a lot of code to enjoy those nice features.

NOTE

Not all Windows Form controls are "databinding aware." When they are aware, they have a DataBindings property.

NOTE

The Data Sources window might end up somewhere else in your IDE. Because your IDE is entirely customizable to your liking, you can customize your windows and tabs to appear wherever you feel they are most productive for you.

Figure 8-16
The Data Sources window

The Car Tracker Application Development

You'll now proceed to the development of the Car Tracker application. First, you need to create a dataset that will provide you with all the databinding you need for the Car Tracker application. Now that your tables are established, you can configure the dataset with all of the elements you've just added to your database.

Before creating a dataset, you must learn what a dataset is. A dataset is an in-memory representation of one or more tables and is used to store the rows you retrieve that match the query you sent to the database. You can then add, delete, or update rows in memory. When the user is done, you can submit, save, or commit the changes to the database. The CarTrackerDataSet.xsd is called an XML Schema Definition file. The .xsd file ensures that the data will be structured and respect the schema. This file will be used later in the project when we discuss databinding.

To create a dataset, you'll learn to use the Data Sources window. This window gives you access to all of the data sources you have configured in your application. See Figure 8-16 to see where the Data Sources window is located. If you don't see the Data Sources window, you can access it by clicking on the Data menu and selecting Show Data Sources. If Show Data Sources does not appear on the Data menu, be sure you have closed all of the CarTracker table data grids and Form1 is visible.

TO CREATE A DATASET

1. In the Data Sources window, click the **Add New Data Source** link or click the **Add New Data Source** button in the toolbar. The Data Source Configuration Wizard appears.

2. The first screen of the Data Source Configuration Wizard allows you to choose the data source type you want to create. You can choose a database, a Web service, or one of your objects. You've just built a database for the Car Tracker application, so you're going to choose that Data Source Type. Select **Database** and then click **Next**.

In the next screen, you will choose your data connection. CarTracker.mdf should already be selected. When you created the CarTracker SQL Server Express database in your project, a data connection was created for you. You can click the **plus sign** (+) in the bottom of the dialog box to see what the connection string looks like. This connection string defines how your application will connect to the database.

3. Click **Next** on the Choose Your Data Connection screen.

The next screen in the wizard inquires whether you want to save this connection string in the application configuration file. As you saw in the previous screen, you know where your database is stored. Yet you might change your mind and deploy the file somewhere else. If you do that, you don't want to modify the source code and recompile it. Putting the connection string in your application configuration file is actually a best practice. It gives you the advantage of only modifying the file and restarting the application without recompilation so as to automatically pick up the changes in your connection string and connect to that new location.

The application configuration is stored in an XML file named using the application's executable name and adding .config at the end of the executable filename. In our application, the file is named CarTracker.exe.config, although you only see app.config while working in Visual Studio. If you want to save the connection string, you are also asked to provide a variable name under which it will be saved in the file.

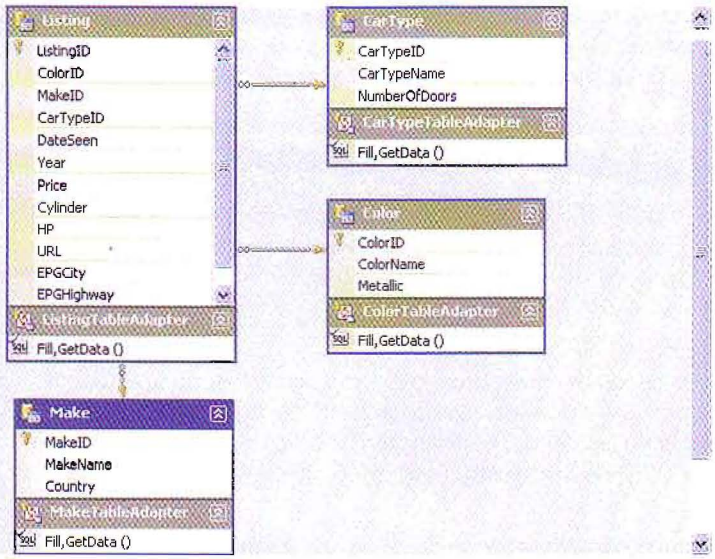
4. Make sure the **Yes, Save The Connection As** check box is selected and then click **Next**.

5. In the next screen, you'll select all the tables from the database that will be in your dataset and name your dataset. In your case, you will need all of the tables, so expand the Tables node and select all tables. Leave the DataSet Name set to CarTrackerDataSet and then click **Finish**.

The result of your dataset configuration is an .xsd file or a XML Schema Document, and it will define the internal structure of your dataset. Remember that a dataset is an in-memory representation of one or more tables from your database. ADO.NET will use this schema file when working with your application. When running the application, the user will be able to add, delete, or modify rows in the dataset (in the computer's memory). The changes will remain in memory until the user commits the changes back to the database, which in our example is the CarTracker.mdf file.

6. In the Solution Explorer, double-click the .xsd file named **CarTrackerDataSet.xsd**. As shown in Figure 8-17, the result of the dataset creation is similar to the database diagram you created earlier. Your diagram might be different depending on your screen resolution and how you customized your IDE.

Figure 8-17
Graphical representation of the CarTracker dataset



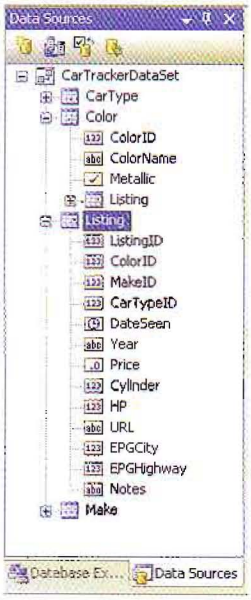


Figure 8-18
View of the Color and Listing dataset tables from within the Data Sources window

There are some notable differences, however. You'll see the same columns that you have created in your physical database, but in the bottom of each table, you will see methods: Fill, GetData(). These methods are particular to the dataset, and the ADO.NET-generated code by Visual Studio will use them to databind data to your Windows Form controls—controls that do not exist yet!

7. Return to the Data Sources window and expand the dataset tables. You'll see the in-memory representation of your tables, and you'll also see that each column has a small icon that gives you its type. These icons may look familiar to you because they are similar to the controls in the toolbox. Refer to Figure 8-18 for a quick glance at the Color and Listing dataset tables and their column types.

8. Close the graphical representation of your dataset by clicking the **X** in the corner of the designer surface.

9. In the Solution Explorer, double-click your **Form1.cs** file to open the designer surface for Form1.

10 In the Data Sources window, select the **Listing** node in your dataset and click on the drop-down arrow point that's next to the word *Listing*. You will be presented with two choices: DataGridView or Details. DataGridView brings all of the dataset fields into a table or grid format with multiple rows, while Details brings the dataset fields in one row at a time with all fields as individual controls. For our example, select **Details**

You'll also see that each member of the dataset has the same drop-down arrow, which allows you to change which controls will be dropped onto the form when it is dragged. Allowing you to choose controls prior to dragging the dataset table onto the form prevents you from having to lay out the UI yourself piece by piece.

11 Change the ColorID, MakeID, and CarTypeID to the ComboBox type by clicking on the drop-down arrow next to each column and selecting **ComboBox**.

12 Select the **Listing** node by clicking on it, and then drag and drop it onto the designer surface on Form1.

13 You'll now modify the form size like you did in previous chapters by modifying the form's Size property. Change the form size so that its **Width** is **450** pixels and its **Height** is **550** pixels.

14 Move all of the controls so that the first label is almost in the top-left corner just beneath the tool strip. See Figure 8-19 to determine how the controls should approximately be placed.

WARNING

You may need to scroll to see all of the controls depending on your screen resolution.

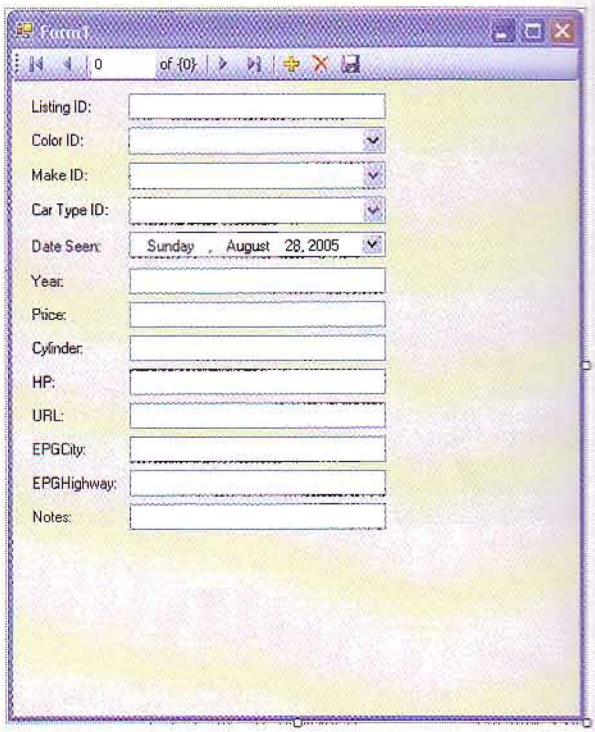


Figure 8-19
Resized Car Tracker form after moving all of the controls

IMPORTANT

When working with local database files, it is necessary to understand that they are treated like any other content file. For desktop projects, it means that, by default, the database file will be copied to the output folder (i.e., bin) each time the project is built. After pressing F5, here's what it would look like on disk:

CarTracker\CarTracker.mdf
CarTracker\Form1.cs
CarTracker\Bin\Debug\CarTracker.mdf
CarTracker\Bin\Debug\CarTracker.exe

At design time, CarTracker\CarTracker.mdf is used by the data tools and wizards. At run time, the application will use the database under the bin\debug folder. As a result of the copy, many people have the impression that the application did not save the data to the database file. This assumption occurs because there are two copies of the data file involved. This also happens when looking at schema/data through the Database Explorer. The tools are using the copy in the project folder and not the file in the bin\debug folder. The following are a few ways to work around this copy behavior:

1. **If you select your database file in the Solution Explorer window, you will see a property called Copy To Output Directory in the Properties window. By default, it is set to Copy Always, which means that data files in the project folder will be copied to the bin\debug folder on each build, thus overwriting the existing data files if any. You can set this property to Do Not Copy and then manually place a copy of the data file in the bin\debug folder. In this way, on subsequent builds, the project system will leave the database file in the bin\debug folder and not try to overwrite it with the one from the project. The downside to this method is that you will still have two copies. Therefore, after you modify the database file using the application, if you want to make those same changes within the project, you will need to copy the changes to the project manually and vice-versa.**
2. **You can leave the data file outside the project and create a connection to it in Database Explorer. When the IDE asks you to bring the file into the project, simply say no. In this way, both the design time and run time will be using the same data file. The downside to this method is that the path in the connection string will be hard coded, and it will therefore be harder to share the project and deploy the application. Before deploying the application, make sure to replace the full path in the settings with a relative path. If you want to read more about the relative path versus the full path (plus a bit more about this copy behavior), read the following article: *http://blogs.msdn.com/smartclientdata/archive/2005/08/26/456886.aspx*. You'll see that I took portions of that article and modified them so that they fit our application.**

As you can see, many things have just happened. Let's start by looking at the designer surface. All of the fields from the dataset have been added as controls, and labels were also added based on the name of the field in the dataset. This feature is called smart caption. Visual Studio uses Pascal or Camel casing as a mechanism to insert a space in labels when using smart captions. When you drop the dataset fields onto the form, smart caption looks at each field's casing. When it finds an uppercase letter or an underscore character (i.e., _) following a lowercase letter, it inserts or replaces the _ with a space. An exception to this rule can be seen in the EPGCity and EPGHighway fields. When you use uppercase letters for an acronym Visual Studio cannot distinguish that these are two words and therefore doesn't split them apart. You'll have to split these two fields manually.

You will also notice that a tool strip has been added that contains almost the same buttons you used while working with the database table designer

15 Read the blue Important sidebar to the left. With this copy behavior in mind, I suggest that you use Approach #1, even though you'll have to perform some manual steps. If you want to debug your application from within Visual Studio, it's preferable to use this solution or you will not be able to see the changes applied to your database file. The database file will always come back to the initial one from your project, which is similar to resetting the whole database to what it is inside Visual Studio

16 Select the **CarTracker.mdf** database file in the Solution Explorer and change the Copy to Output Directory property to **Do Not Copy** in the Properties window.

17 Press **F5** to build and run your application. You'll get an exception message because the file won't be copied in the bin\debug directory. Also, on the form load event when your code tries to fill the dataset, it won't find the database at the place specified by the connection string. Therefore, you get an SQLException stating that it's not able to attach to the database. Click the **Stop Debugging** button or press **Shift+F5** to stop debugging

18 Using Windows Explorer, go into your project directory (it should be located at My Documents\Visual Studio 2005\Projects\CarTracker\CarTracker\) and copy the **.mdf** and **.ldf** files into the bin\debug directory under CarTracker

19 In Visual C# Express, press **F5** to build and run your application again.

You should see the two records that you've inserted manually into the Listing table. You should be able to navigate using the tool strip and also modify, insert, and delete a record. See Figure 8-20 for a snapshot of your Car Tracker application at run time.

20 Change the URL of the row at position 1 to end with **.net** instead **.com**.

21 After changing the URL for the record, press the **disk icon** to commit the changes to the database.

22 Close the Car Tracker application and restart it by pressing **F5**. You should now see the first row with the modified URL ending in .net. Close the application again.

23 To verify that you are working with a design time and a run time version of the CarTracker database, open the **Listing** table and select **Show Table Data** from within the Database Explorer. The first row should contain a URL column ending in .com and *not* in .net. Point proven! The database file in Visual Studio is now de-coupled with the one your application is using at run time. Read the More Info note on the next page to learn how to make the data the same in both the design and run time.

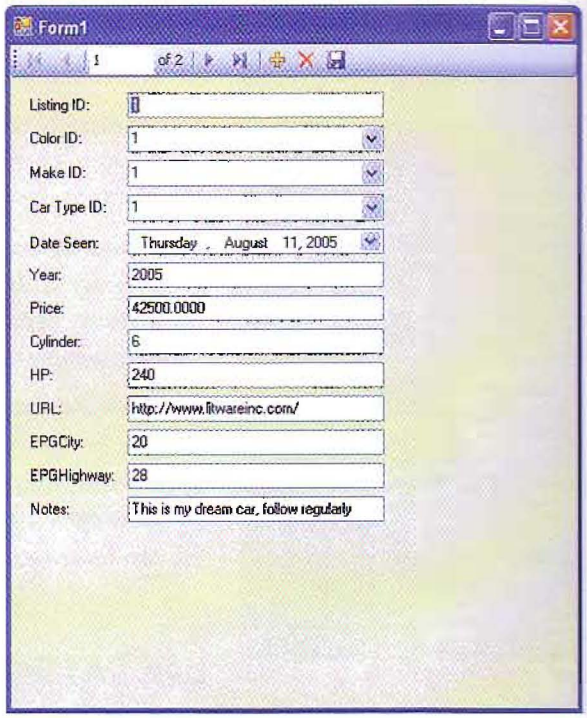


Figure 8-20
Execution of the Car Tracker application

Component Tray

When you dragged the Listing dataset table onto the designer surface, you probably saw that four items were added in the gray area below the designer surface. This section of the designer surface is called the component tray. This is the section that Visual Studio uses for nonvisual controls. In your case, it added an instance of the CarTracker dataset, a Listing table adapter, a Listing binding source, and finally a Listing binding navigator.

Let's describe them individually.

- **Binding Source** You can think of a binding source as a "broker" or a layer of indirection.

MORE INFO

If you want the same data in Visual Studio as you have when executing the application in debug mode, then you must close your project completely. Using Windows Explorer, copy the .mdf and .ldf files from the bin\debug folder to the project folder. When you reopen your project, the database will now contain the same content.

Suppose you then want to modify the structure of your database, such as adding a column to a table. If you don't want to lose the data within the bin\debug database files, you must copy them back to the project folder before you modify the table structure. When done with the modifications, you simply copy both the .mdf and .ldf files back to the bin\debug folder. Of course, if your application needs those new database changes, you would also have to modify the dataset, but that process is beyond the scope of this book.

It can also be viewed as an intermediary between a data-bound control on your form and a data source, such a a dataset. A binding source provides currency management and notifications services (events). The binding source has many methods to facilitate, such as sorting, filtering, navigating, and editing of data from its data-bound controls to the data source. It's also linked tightly to the next component: the binding navigator. When you see a binding navigator, you're assured of getting a binding source.

- **Binding Navigator** The binding navigator is a means to enable navigation and data manipulation. It has a UI component or, more specifically, a tool strip with buttons to facilitate the functionalities provided by the binding source.
- **Typed Dataset** Although you know what a dataset is, you may not know that it's a strongly typed object. It contains data tables of the DataTable type that constitute the in-memory representation of your database tables. These data tables also have a special data adapter called the table adapter. There is a table adapter for each data table.
- **Table Adapter** A table adapter is a data access object. It connects to the database (e.g., SQL Server 2005 Express Edition), executes the queries, and fills a data table with data when it returns from SQL Server. Therefore, it's the central point for all data access on an individual table. There is one table adapter per table in your data source. A table adapter can have more than one SELECT query.

How Do I Get More Meaningful Information on My Form?

Let's return to our CarTracker project. As you can see, the ColorID, MakeID, and CarTypeID combo boxes are there, but they are displaying the ID and not the name associated with the ID. This is not helpful for the user because an ID doesn't have any meaning to the user and he/she might not be able to easily add or modify rows without having a human-readable format for those columns. Consequently, you need to make sure the data is displayed in a humanly readable way and that the ID is stored in the row whenever the user modifies the information.

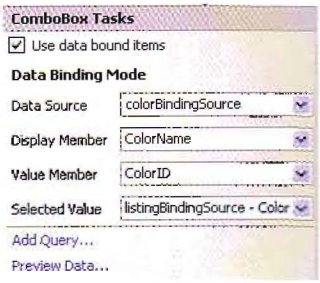
There's an easy way to accomplish this, which you will do now for your three combo boxes.

TO DATABIND WITH DOMAIN TABLES

1. In the Data Sources window, select the **Color** table from the dataset, drag it onto the form's designer surface over the ColorID combo box, and drop it

You'll see that another table adapter (**colorTableAdapter**) and another binding source (**colorBindingSource**) were added to the component tray. If you go to the ColorID combo box and click on the smart tag triangle, you'll see the Data Binding Mode information box appear, as shown in Figure 8-21. You'll notice that your drag-and-drop action bound the combo box control with the ColorBindingSource. Because of this action, whenever the combo box displays, it will display the color names instead of ColorID. When the user picks a color from the combo box, the associated value member that will be used in the row will still be the ColorID, specifically, the ColorID associated with the ColorName. Wonderful, isn't it? And no lines of code were used.

Figure 8-21
ColorID combo box smart tag information showing Data Binding Mode



2. Repeat the same process for the Make and CarType dataset tables and the corresponding MakeID and CarTypeID combo boxes.

3. Build and run your application and then look at each combo box. You now have real color names and not merely ColorIDs; the same is true for CarType and Make. The combo boxes are also populated with all of the values coming from those tables and not simply the value for that specific row. Click on the drop-down arrow and you'll see all other potential values. Close the application.

4. On the form, remove the "ID" part from the **Color ID**, **Make ID**, and **Car Type ID** labels.

5. You will now enlarge the Notes field by making it a multi-line text box. Select the **Notes** text box and change the Multiline property to **true**. Also change the **MaxLength** to **250**, the **Size:Height** to **50**, and the **Size:Width** to **250**.

6. Delete the **ListingID** text box and its label.

MORE INFO

This intelligent databinding is a new Visual Studio feature called Smart Defaults. Smart Defaults looks in the dataset table to see whether there's a column of type string by either the ID or the primary key. If so, it tries to use this one for the databinding.

7. Size and reposition the controls on the form so that it resembles the form shown in Figure 8-22; it does not need to be an exact duplicate. It will be good practice to bring back UI design concepts from Chapter 5 and also good preparation for Chapter 9. Change the Text property of the form to "Car Tracker".

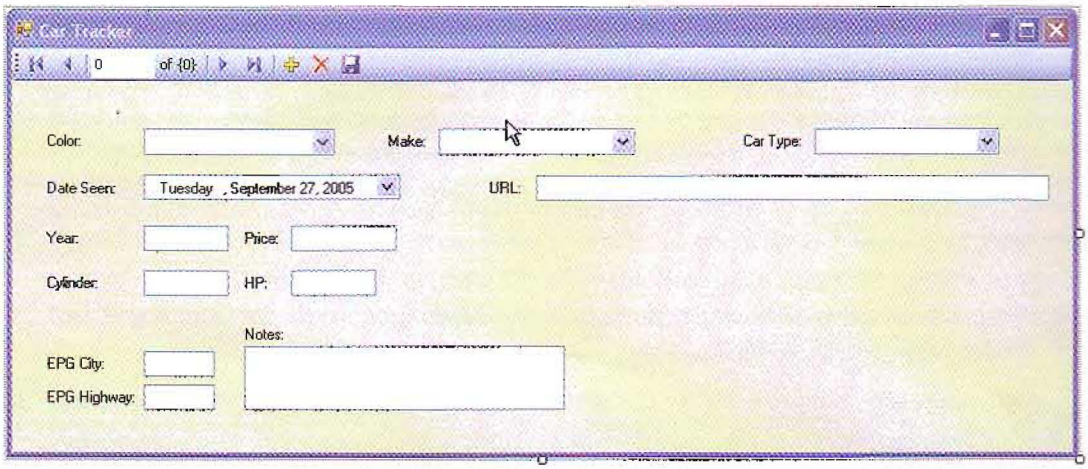


Figure 8-22
New visual aspects of the Car Tracker application

8. In the Solution Explorer, rename form1.cs to **Main.cs**. When asked if you want to rename all references in this project, click **Yes**.

9. Select the form and change the BackColor property to **Highlight**.

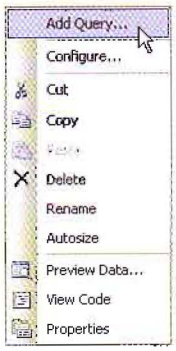
Everything is nearly complete for this application, but the research capabilities are lacking. Currently, the only way to search is to scan through all of the rows until you find the correct one. This is not difficult now because you have only two rows in your Car Tracker database. Yet, if you had 500 rows, the Scan method would not be effective at all! Therefore, you'll implement search capabilities by adding queries to your application by using the Dataset Designer

10. In the Data Sources window, select **CarTrackerDataSet**. Right-click and select **Edit DataSet with Designer**.

11. Select the **Listing** data table, and then select the **ListingTableAdapter** section at the bottom of the data table.

When you look in the Properties window, you'll see that four types of queries were automatically generated by Visual Studio. SELECT, INSERT, DELETE, and UPDATE. They are the queries that helped you have a fully workable application without writing a single line of code. When you read about table adapters earlier, you learned that you can have multiple queries with a table adapter because it is the central point of data access. You will thus add search capabilities to your application by adding queries to the table adapters and by using parameters from the UI. First, you will add the ability to search for listings that have a certain color.

Figure 8-23
Adding new queries to a table adapter



12 Right-click the **ListingTableAdapter** section and select **Add Query. . .** as shown in Figure 8-23.

This will bring you to the TableAdapter Query Configuration Wizard. This wizard will help you add another SELECT query that will use parameters to refine your search. You can also create a SELECT query and turn it into a stored procedure or use an existing stored procedure. As its name implies, a stored procedure is one that is stored in SQL Server and contains SQL statements, along with other programming constructs, that use a language called T-SQL or Transact-SQL. A new feature to SQL Server 2005 Express Edition is that stored procedures can also be coded in managed languages, such as C# and Visual Basic. Stored procedures are executed on the server. Since you're using SQL Server 2005 Express Edition, this will be of no concern because the SQL Server and the application are executed on the same machine.

13 Select **Use SQL statements** and click **Next**. When asked which type of SQL query you want to use, choose **SELECT which return rows** and then click **Next**. Note that you could have added any SQL query type you wanted.

14 You are now presented with an edit window in which to add your SQL statement that will perform a search for all of the listings that contain a particular color. Refer to Figure 8-24 to see the SQL command edit window. Click on the **Query Builder. . .** button to get a visual view of the query.

15 You will now add the Color table to the diagram so that you'll be able to base your search on a particular color. To add the Color table, simply right-click in the diagram area and select **Add Table. . .** The Add Table dialog box appears as shown in Figure 8-25. Select the **Color** table and click the **Add** button. When the Color table has been added to the diagram, click the **Close** button.

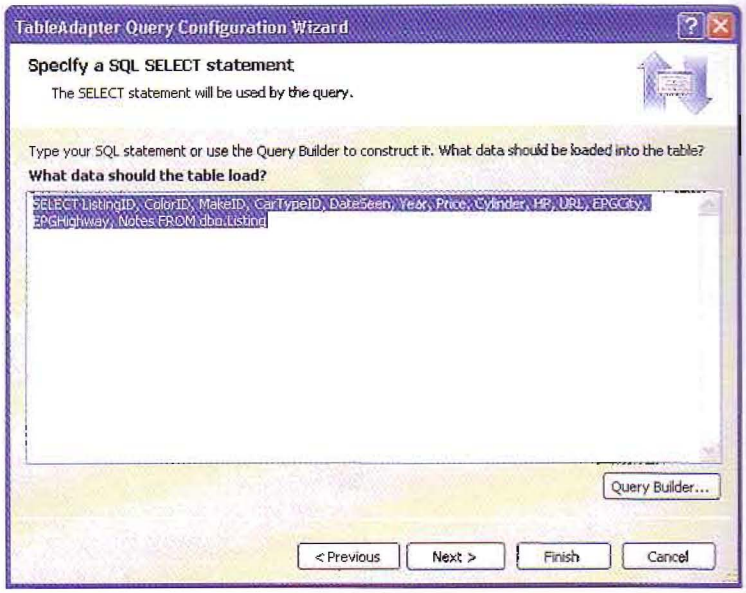
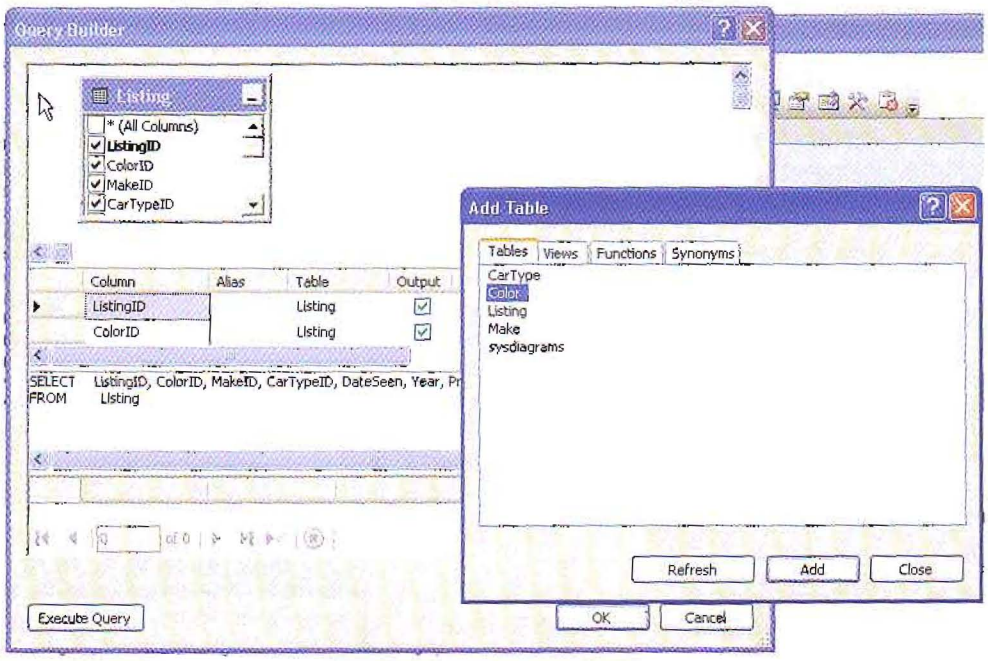


Figure 8-24
SQL command edit window ready to customize the user's search

Figure 8-25
The Add Table dialog box



MORE INFO

The '%' symbol is the wildcard in SQL and it can mean anything. For example, in the previous WHERE clause, it means return something that has a color similar to the colorname parameter.

16 In the SQL code pane of Query Builder, append the following SQL code that will help in the filtering process:

```
WHERE      (Color.ColorName LIKE '%' + @colorname + '%')
```

17 Before you proceed with your new query, make sure it will give you the results you're expecting. Click the **Execute Query** button to display the Query Parameters dialog box, as shown in Figure 8-26.

18 Try replacing the word *NULL* with **blue** and then click **OK**. The Results pane of Query Builder should display only one row. Using the word **black** should return the black car row. Simply enter the letter **b**, and you should get both the blue and the black rows. Once you're satisfied with your query, click **OK** in Query Builder.

Figure 8-26
Query Parameters dialog box with prompt to enter a color name value

19 On the Specify a SQL SELECT Statement screen of the wizard, click **Next**. It's time to add your query to the application.

20 A screen appears that will prompt you to name the methods that your query will generate. Those methods will be available after this creation from the Listing table adapter. Refer to Figure 8-27 to view this screen containing the two new method names. For both names you basically need to add what your filter is. In your case, you can add ColorName since you filtered by that in your WHERE clause. When done, click **Next**.

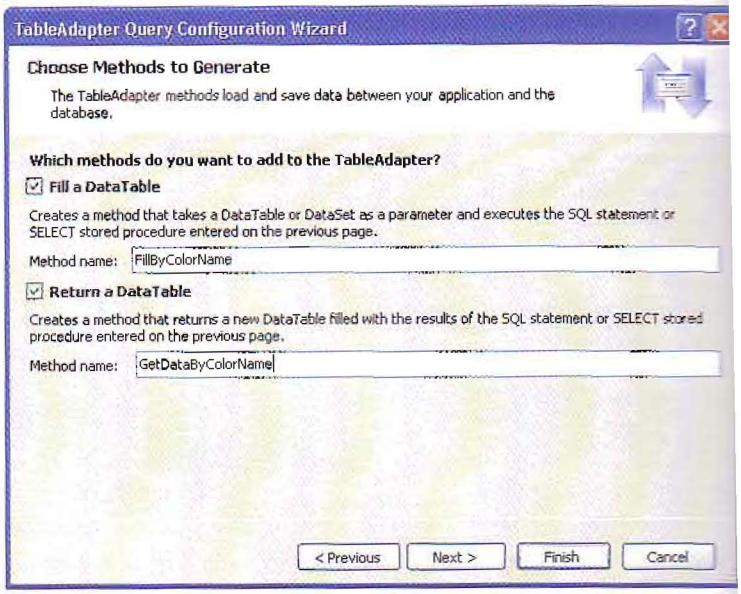


Figure 8-27
Dialog box with prompt to rename the methods used to increase search capabilities

21 After processing for a few seconds, the computer should come back with a results screen informing you that your Select statement and your new Fill and Get methods are ready to use. Click the **Finish** button.

Look at the table adapter section of the Listing data table. Your new methods will be added there.

22 Repeat steps starting at step 12 and add another query to the ListingTableAdapter for the CarType table. Use the following WHERE clause:

```
WHERE (CarType.CarTypeName LIKE '%' + @cartypename + '%')
```

Name the fill and get data methods **FillByCarTypeName** and **GetDataByCarTypeName**.

23 Repeat steps starting at step 12 and add another query to the ListingTableAdapter for the Make table. Use the following WHERE clause:

```
WHERE (Make.MakeName LIKE '%' + @makename + '%')
```

Name the fill and get data methods **FillByMakeName** and **GetDataByMakeName**.

You're almost done now, but you still have to bind those new queries to controls on your form. On any data-bound control on your form, you can select **Add Query...** from the smart tag menu. In your case, you want to add search capabilities to the whole form and not simply to a particular control.

24 Go to the component tray, click the **ListingTableAdapter** smart tag, and select **Add Query. . .** You'll see a Search Criteria Builder dialog box that will prompt you to create a new query or pick an existing one. Since you just built three new sets of methods, you merely need to select one. Select the **Existing Query Name** option and then select **FillByColorName** as shown in Figure 8-28.

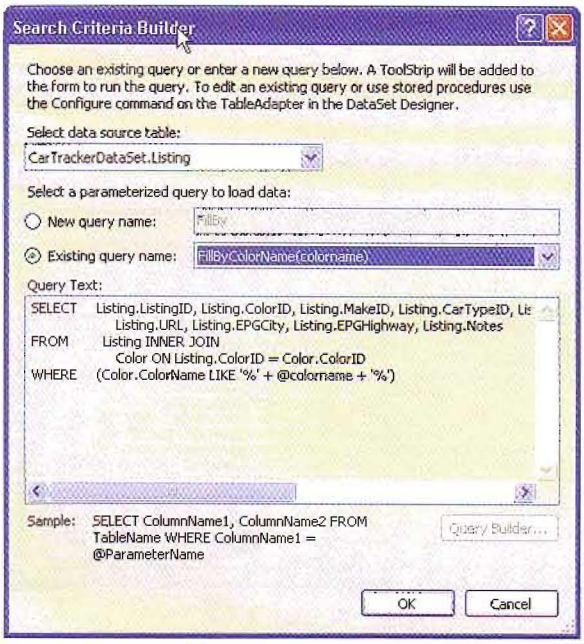


Figure 8-28
Search Criteria Builder with the FillByColorName method selected

25 Click the **OK** button. You'll see that a tool strip has been placed at the top of the form with a search button that will call your method when you click it, thereby giving you a way of searching by certain criteria. This was accomplished by only typing in the three WHERE clauses for your specific queries.

26 Repeat step 24 and add the **FillByCarTypeName** and **FillByMakeName** queries, which will add two more tool strips.

27 Now add a tool strip container to the form like you did in Chapter 6. Set the Dock property to fill the form. In the smart tag menu, select **Re-Parent Controls** to place all of your tool strips on the top panel and all of your other controls in your content panel. If necessary, use the Document Outline window to view and adjust the hierarchy of objects on the form.

28 When you're through moving controls, extend the top panel by clicking the grip and pulling it down so that it becomes two tool strips wide.

29 Make sure your application looks like the one shown in Figure 8-29. Press **F5** to see the results of your work. Type "blue" in the colorname tool strip and click **FillByColorName** to see if it returns blue color car listings. Experiment with the other features of the application.

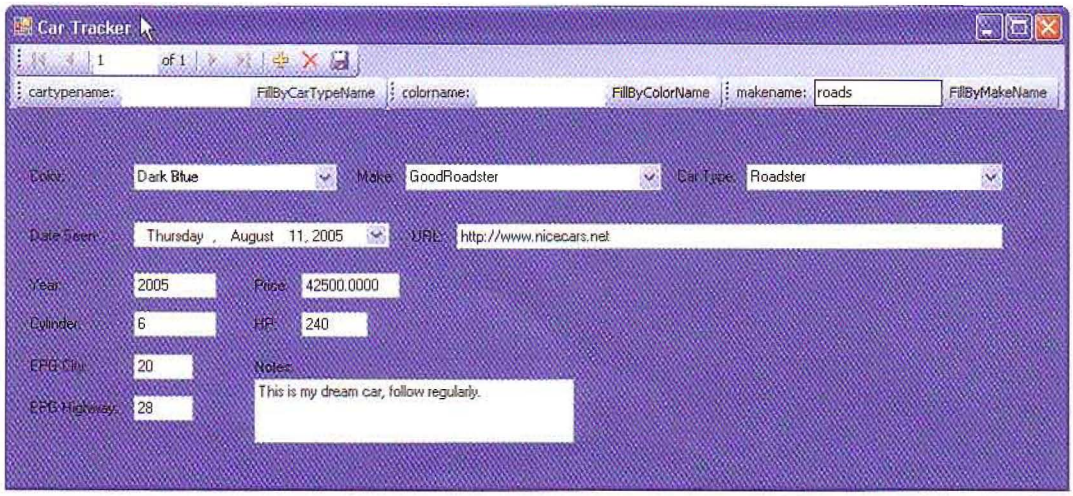


Figure 8-29
Final Car Tracker application screen

This was a simple application that you can probably modify to handle more information, such as car pictures. But there is nothing that you can't add by yourself now! Here's a list of other things you can do if you want to continue to work on this application

- Add validations for user input, such as making sure the year of the car is not greater than the current year + 2
- Add pictures in the databases and on the form
- Add a sold check mark
- Add three forms to add data in the domain tables (CarType, Make, Color)
- Add more information in the listing, such as contact information
- Make the URL clickable
- Save an ad as a text file

In Summary...

That was a big chapter with a lot of material! Let's review what you've learned. You were first introduced to databases and database concepts. You learned what constituted a database and what you usually find within a database. You learned about data integrity and how it is related to the primary key and the foreign key.

You then used Visual C# 2005 Express Edition to create a database and tables and then populated them with some initial data using various tools in Visual C# 2005 Express Edition. You implemented all of the foreign key relationships without leaving Visual Studio and validated them as well.

After entering your data manually, you learned how to allow a user to enter data more easily by developing a sample Car Tracker application that uses ADO.NET and databinding.

Lastly, you learned about the new components of ADO.NET 2.0 and how, with little or no code, you can develop a fully working data-centric application. You've only been introduced to a brief part of ADO.NET, for it's a vast subject. If you want to learn more, try looking at some code or samples on MSDN. A good place to begin is the 101 samples for Visual Studio 2005. Pay particular attention to the data access and Windows Forms samples. Here's the link: *http://go.microsoft.com/fwlink/?linkid=51659*.

In the next chapter, you will develop the final application of this book—the Weather Tracker application. You'll learn new concepts such as deployment, consuming Web services, user settings, and much more in a complete application with all of the necessary validations.