In Chapter 14, you saw how to use data readers to access database data in a connected, forward-only, read-only fashion. Often, this is all you want to do, and a data reader suits your purposes perfectly.

In this chapter, you’ll look at a new object for accessing data, the *data set*. Unlike data readers, which are objects of data provider–specific classes that implement the `System.Data.IDataReader` interface, data sets are objects of the class `System.Data.Dataset`, a distinct ADO.NET component used by all data providers. Data sets are completely independent of and can be used either connected to or disconnected from data sources. Their fundamental purpose is to provide a relational view of data stored in an in-memory cache.

**Note** In yet another somewhat confusing bit of terminology, the class is named `DataSet`, but the generic term is spelled *data set*.

So, if a data set doesn’t have to be connected to a database, how do you populate it with data and save its data to the database? This is where *data adapters* come in. Think of data adapters as bridges between data sets and data sources. Without a data adapter, a data set can’t access any kind of data source. The data adapter takes care of all connection details for the data set, populates it with data, and updates the data source.

In this chapter, we’ll cover the following:

- Understanding the object model
- Working with data sets and data adapters
- Propagating changes to a data source
- Concurrency
- Using data sets and XML
- Understanding typed and untyped data sets

**Understanding the Object Model**

We’ll start this chapter with a quick presentation of all the new objects you’ll need to understand in order to work with data sets and data adapters. You’ll start by looking at the difference between data sets
and data readers and then move on to look in more detail at how data is structured within a data set and how a data set works in collaboration with a data adapter.

Data Sets vs. Data Readers

If you simply want to read and display data, then you need to use only a data reader, as you saw in the previous chapter, particularly if you're working with large quantities of data. In situations where you need to loop through thousands or millions of rows, you want a fast sequential reader (reading rows from the result set one at a time), and the data reader does this job in an efficient way.

If you need to manipulate the data in any way and then update the database, you need to use a data set. A data adapter fills a data set by using a data reader; additional resources are needed to save data for disconnected use. You need to think about whether you really need a data set; otherwise, you'll just be wasting resources. Unless you need to update the data source or use other data set features such as reading and writing to XML files, exporting database schemas, and creating XML views of a database, you should use a data reader.

A Brief Introduction to Data Sets

The notion of a data set in ADO.NET is a big step in the world of multitiered database application development. When retrieving or modifying large amounts of data, maintaining an open connection to a data source while waiting for users to make requests is an enormous waste of precious resources.

Data sets help tremendously here, because they enable you to store and modify large amounts of data in a local cache, view the data as tables, and process the data in an offline mode (in other words, disconnected from the database).

Let's look at an example. Imagine you're trying to connect to a remote database server over the Internet for detailed information about some business transactions. You search on a particular date for all available transactions, and the results are displayed. Behind the scenes, your application creates a connection with the data source, joins a couple of tables, and retrieves the results. Suppose you now want to edit this information and add or remove details. Whatever the reason, your application will go through the same cycle over and over again: creating a new connection, joining tables, and retrieving data. Not only is there overhead in creating a new connection each time, but you may be doing a lot of other redundant work, especially if you're dealing with the same data. Wouldn't it be better if you could connect to the data source once, store the data locally in a structure that resembles a relational database, close the connection, modify the local data, and then propagate the changes to the data source when the time is right?

This is exactly what the data set is designed to do. A data set stores relational data as collections of data tables. You met data tables briefly in the previous chapter when a System.Data.DataTable object was to hold schema information. In that instance, however, the data table contained only schema information, but in a data set, the data tables contain both metadata describing the structure of the data and the data itself.