Databases and LINQ

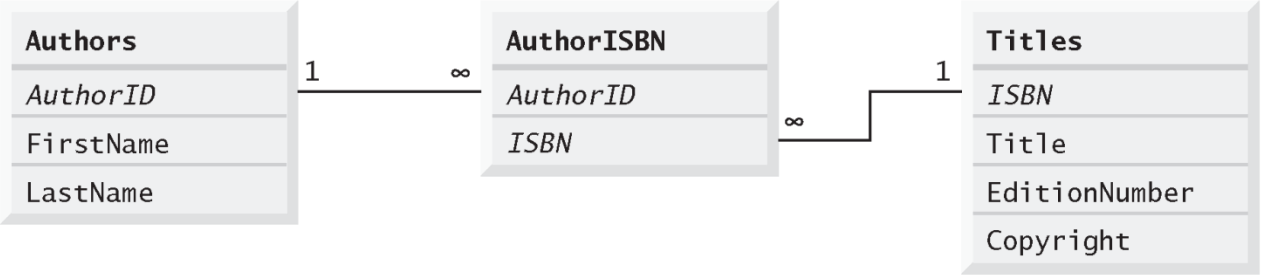


Fig. 22.9 | Entity-relationship diagram for the Books database.

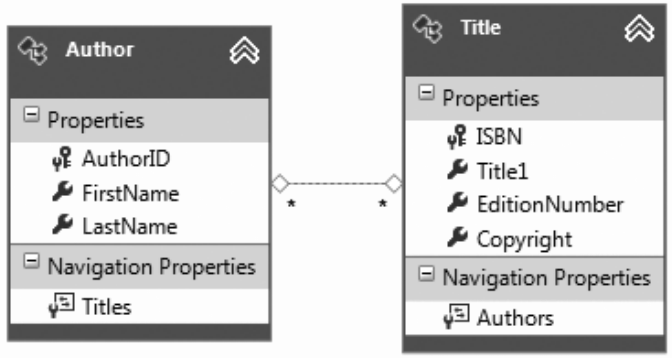


Fig. 22.16 | Entity data model diagram for the **Author** and **Title** entities.

```
// Fig. 22.20: DisplayAuthorsTable.cs
// Displaying data from a database table in a DataGridView.
using System;
using System.Data.Entity;
using System.Data.Entity.Validation;
using System.Linq;
using System.Windows.Forms;

namespace DisplayTable
{
   public partial class DisplayAuthorsTable : Form
   {
      // constructor
      public DisplayAuthorsTable()
      {
         InitializeComponent();
      } // end constructor

      // Entity Framework DbContext
      private BooksExamples.BooksEntities dbcontext =
         new BooksExamples.BooksEntities();

```

Fig. 22.20 | Displaying data from a database table in a `DataGridView`. (Part 1 of 4.)

```
        // load data from database into DataGridView
        private void DisplayAuthorsTable_Load( object sender, EventArgs e )
        {
           // load Authors table ordered by LastName then FirstName
           dbcontext.Authors
              .OrderBy( author => author.LastName )
              .ThenBy( author => author.FirstName )
              .Load();

           // specify DataSource for authorBindingSource
           authorBindingSource.DataSource = dbcontext.Authors.Local;
        } // end method DisplayAuthorsTable_Load

        // click event handler for the Save Button in the
        // BindingNavigator saves the changes made to the data
        private void authorBindingNavigatorSaveItem_Click(
           object sender, EventArgs e )
        {
           Validate(); // validate the input fields
           authorBindingSource.EndEdit(); // complete current edit, if any

```

Fig. 22.20 | Displaying data from a database table in a `DataGridView`. (Part 2 of 4.)

```
            // try to save changes
            try
            {
               dbcontext.SaveChanges(); // write changes to database file
            } // end try
            catch( DbEntityValidationException )
            {
               MessageBox.Show( "FirstName and LastName must contain values",
                  "Entity Validation Exception" );
            } // end catch
         } // end method authorBindingNavigatorSaveItem_Click
      } // end class DisplayAuthorsTable
   } // end namespace DisplayTable
```

Fig. 22.20 | Displaying data from a database table in a `DataGridView`. (Part 3 of 4.)

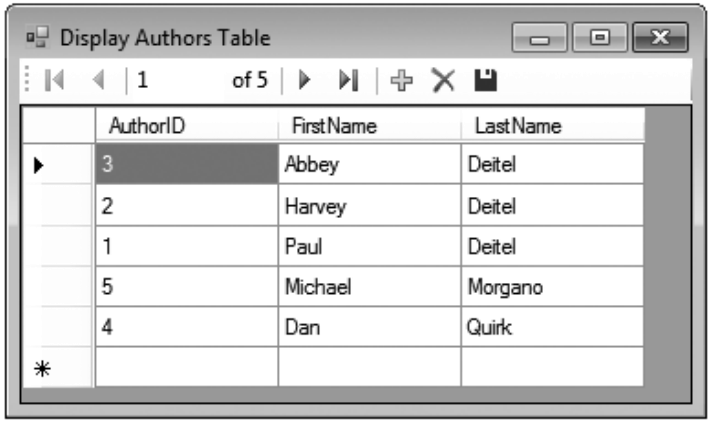


Fig. 22.20 | Displaying data from a database table in a `DataGridView`. (Part 4 of 4.)

a) Results of the "All titles" query, which shows the contents of the `Titles` table ordered by the book titles

Display Query Results

1 of 9

ISBN	Title1	EditionNumber	Copyright
0132121360	Android for Programmers: An App-Driven Approach	1	2012
013299044X	C How to Program	7	2013
0133378713	C++ How to Program	9	2014
0132151006	Internet & World Wide Web How to Program	5	2012
0132575663	Java How to Program	9	2012
0132990601	Simply Visual Basic 2010	4	2013
0133406954	Visual Basic 2012 How to Program	6	2014
0133379337	Visual C# 2012 How to Program	5	2014
0136151574	Visual C++ 2008 How to Program	2	2008

All titles

b) Results of the "Titles with 2014 copyright" query

Display Query Results

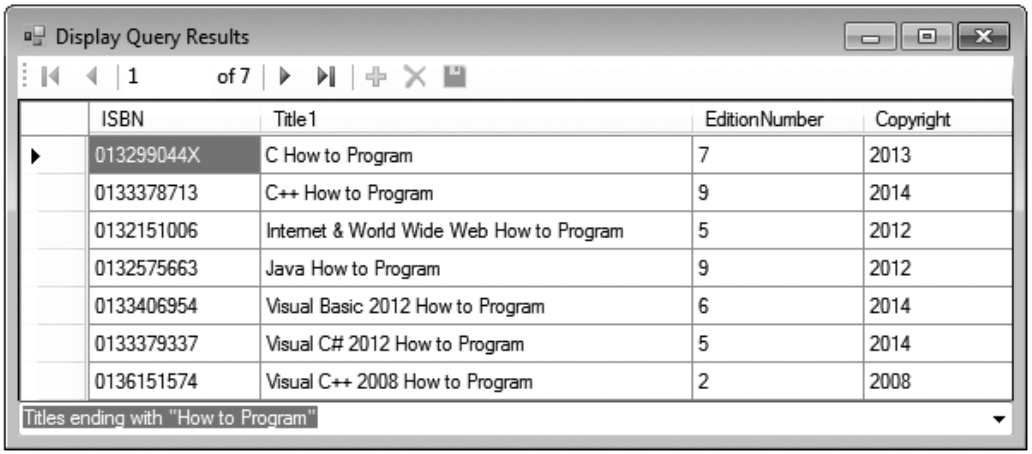
1 of 3

ISBN	Title1	EditionNumber	Copyright
0133378713	C++ How to Program	9	2014
0133406954	Visual Basic 2012 How to Program	6	2014
0133379337	Visual C# 2012 How to Program	5	2014

Titles with 2014 copyright

Fig. 22.21 | Sample execution of the **Display Query Results** app. (Part 1 of 2.)

c) Results of the "Titles ending with 'How to Program'" query



ISBN	Title1	EditionNumber	Copyright
013299044X	C How to Program	7	2013
0133378713	C++ How to Program	9	2014
0132151006	Internet & World Wide Web How to Program	5	2012
0132575663	Java How to Program	9	2012
0133406954	Visual Basic 2012 How to Program	6	2014
0133379337	Visual C# 2012 How to Program	5	2014
0136151574	Visual C++ 2008 How to Program	2	2008

Fig. 22.21 | Sample execution of the **Display Query Results** app. (Part 2 of 2.)

```
// Fig. 22.22: TitleQueries.cs
// Displaying the result of a user-selected query in a DataGridView.
using System;
using System.Data.Entity;
using System.Linq;
using System.Windows.Forms;

namespace DisplayQueryResult
{
   public partial class TitleQueries : Form
   {
      public TitleQueries()
      {
         InitializeComponent();
      } // end constructor

      // Entity Framework DbContext
      private BooksExamples.BooksEntities dbcontext =
         new BooksExamples.BooksEntities();

```

Fig. 22.22 | Displaying the result of a user-selected query in a `DataGridView`. (Part 1 of 3.)

```
        // load data from database into DataGridView
        private void TitleQueries_Load( object sender, EventArgs e )
        {
           dbcontext.Titles.Load(); // load Titles table into memory

           // set the ComboBox to show the default query that
           // selects all books from the Titles table
           queriesComboBox.SelectedIndex = 0;
        } // end method TitleQueries_Load

        // loads data into titleBindingSource based on user-selected query
        private void queriesComboBox_SelectedIndexChanged(
           object sender, EventArgs e )
        {
           // set the data displayed according to what is selected
           switch ( queriesComboBox.SelectedIndex )
           {
              case 0: // all titles
                 // use LINQ to order the books by title
                 titleBindingSource.DataSource =
                    dbcontext.Titles.Local.OrderBy( book => book.Title1 );
                 break;
```

Fig. 22.22 | Displaying the result of a user-selected query in a `DataGridView`. (Part 2 of 3.)

```
            case 1: // titles with 2014 copyright
               // use LINQ to get titles with 2014
               // copyright and sort them by title
               titleBindingSource.DataSource =
                  dbcontext.Titles.Local
                     .Where( book => book.Copyright == "2014" )
                     .OrderBy( book => book.Title1 );
               break;
            case 2: // titles ending with "How to Program"
               // use LINQ to get titles ending with
               // "How to Program" and sort them by title
               titleBindingSource.DataSource =
                  dbcontext.Titles.Local
                     .Where( book =>
                        book.Title1.EndsWith( "How to Program" ) )
                     .OrderBy( book => book.Title1 );
               break;
         } // end switch

         titleBindingSource.MoveFirst(); // move to first entry
      } // end method queriesComboBox_SelectedIndexChanged
   } // end class TitleQueries
} // end namespace DisplayQueryResult
```

Fig. 22.22 | Displaying the result of a user-selected query in a `DataGridView`. (Part 3 of 3.)

a) List of authors and the ISBNs of the books they've authored; sort the authors by last name then first name

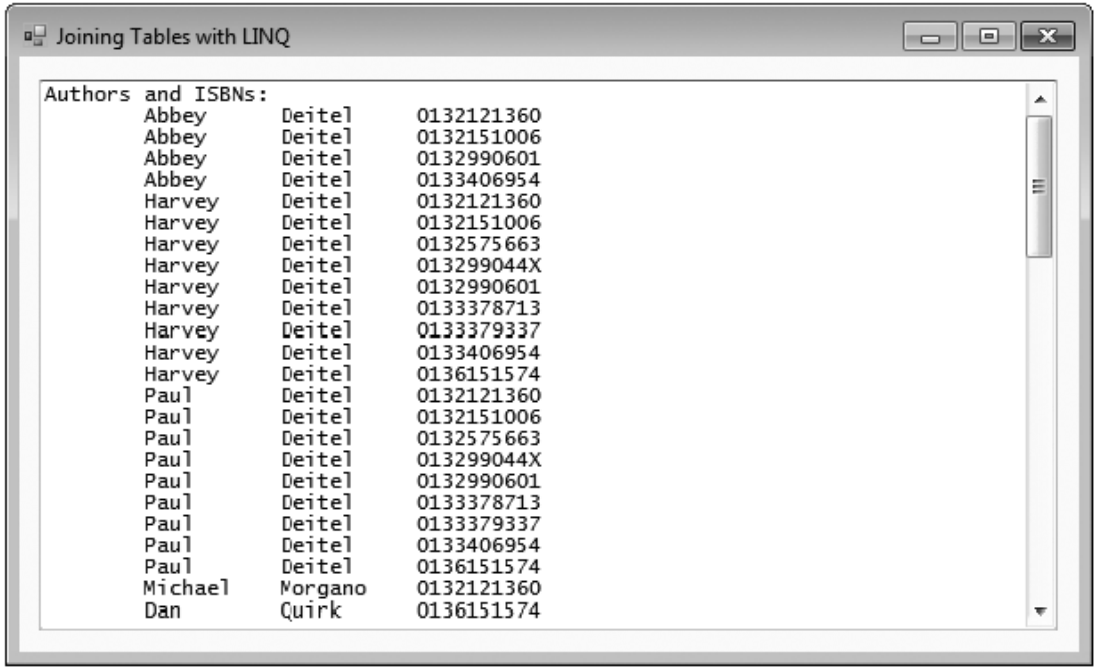


Fig. 22.23 | Outputs from the **Joining Tables with LINQ** app. (Part 1 of 3.)

b) List of authors and the titles of the book's they've authored; sort the authors by last name then first name; for a given author, sort the titles alphabetically

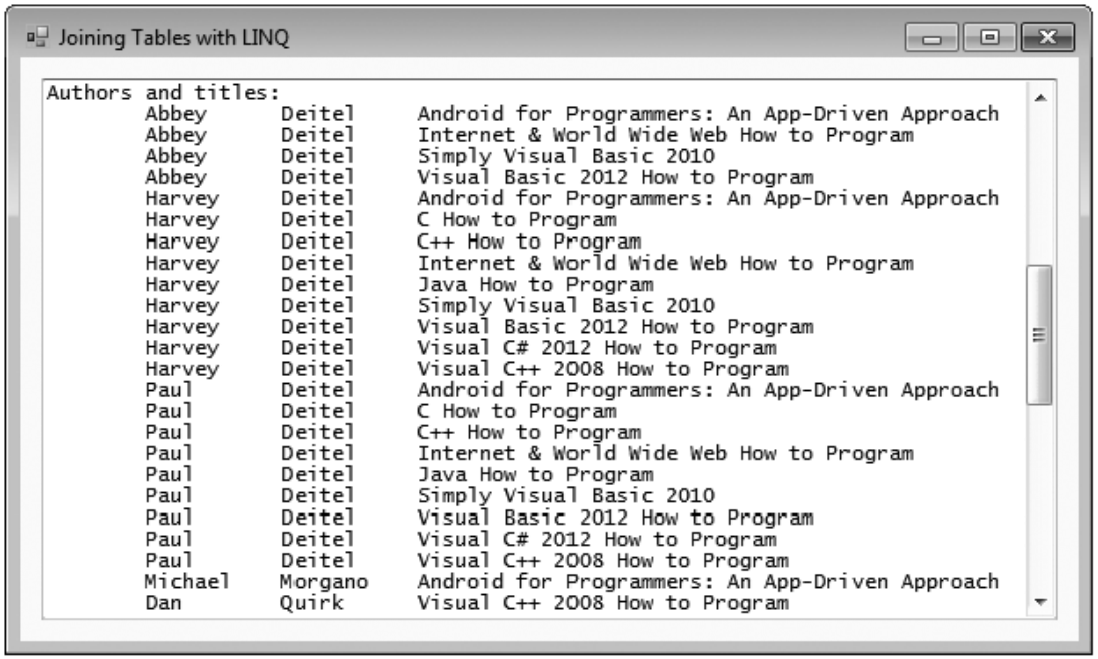


Fig. 22.23 | Outputs from the **Joining Tables with LINQ** app. (Part 2 of 3.)

c) List of titles grouped by author; sort the authors by last name then first name; for a given author, sort the titles alphabetically

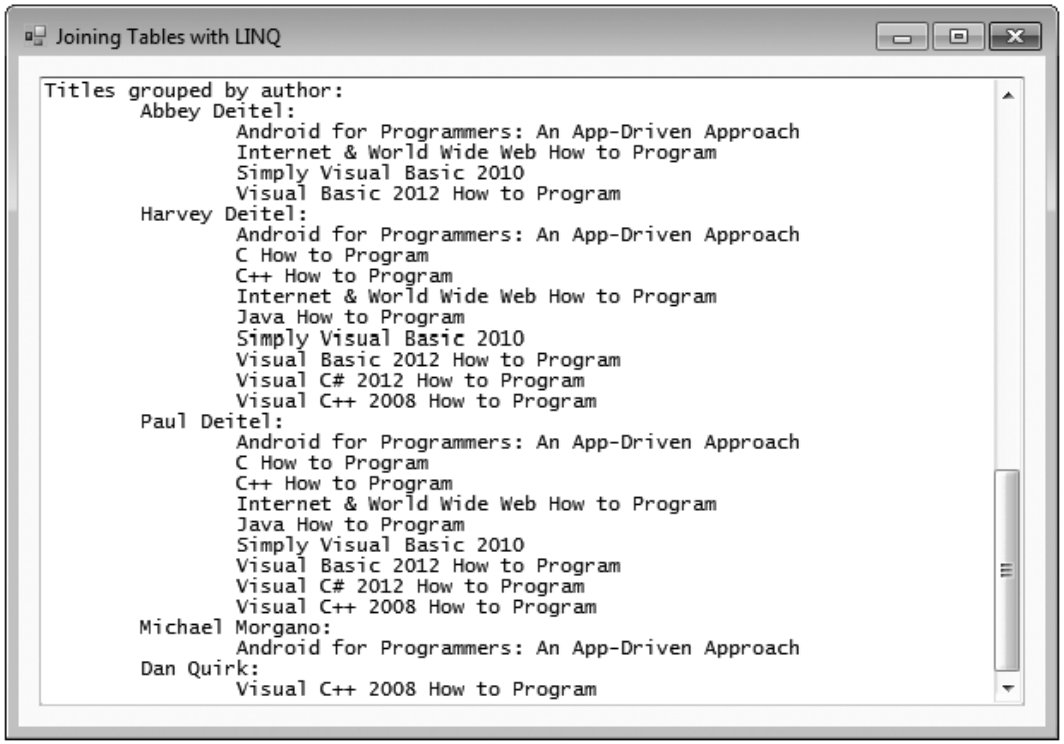


Fig. 22.23 | Outputs from the **Joining Tables with LINQ** app. (Part 3 of 3.)

```
// Fig. 22.24: JoiningTableData.cs
// Using LINQ to perform a join and aggregate data across tables.
using System;
using System.Linq;
using System.Windows.Forms;

namespace JoinQueries
{
   public partial class JoiningTableData : Form
   {
      public JoiningTableData()
      {
         InitializeComponent();
      } // end constructor

      private void JoiningTableData_Load(object sender, EventArgs e)
      {
         // Entity Framework DbContext
         BooksExamples.BooksEntities dbcontext =
            new BooksExamples.BooksEntities();

```

Fig. 22.24 | Creating the `BooksDataContext` for querying the `Books` database.

```
            // get authors and ISBNs of each book they co-authored
            var authorsAndISBNs =
               from author in dbcontext.Authors
               from book in author.Titles
               orderby author.LastName, author.FirstName
               select new { author.FirstName, author.LastName, book.ISBN };

            outputTextBox.AppendText( "Authors and ISBNs:" );

            // display authors and ISBNs in tabular format
            foreach ( var element in authorsAndISBNs )
            {
               outputTextBox.AppendText(
                  String.Format( "\r\n\t{0,-10} {1,-10} {2,-10}",
                     element.FirstName, element.LastName, element.ISBN ) );
            } // end foreach

```

Fig. 22.25 | Getting a list of authors and the ISBNs of the books they've authored.

```
            // get authors and titles of each book they co-authored
            var authorsAndTitles =
               from book in dbcontext.Titles
               from author in book.Authors
               orderby author.LastName, author.FirstName, book.Title1
               select new { author.FirstName, author.LastName,
                  book.Title1 };

            outputTextBox.AppendText( "\r\n\r\nAuthors and titles:" );

            // display authors and titles in tabular format
            foreach ( var element in authorsAndTitles )
            {
               outputTextBox.AppendText(
                  String.Format( "\r\n\t{0,-10} {1,-10} {2}",
                     element.FirstName, element.LastName, element.Title1 ) );
            } // end foreach

```

Fig. 22.26 | Getting a list of authors and the titles of the books they've authored.

```
            // get authors and titles of each book
            // they co-authored; group by author
            var titlesByAuthor =
               from author in dbcontext.Authors
               orderby author.LastName, author.FirstName
               select new { Name = author.FirstName + " " + author.LastName,
                  Titles =
                     from book in author.Titles
                     orderby book.Title1
                     select book.Title1 };

            outputTextBox.AppendText( "\r\n\r\nTitles grouped by author:" );

```

Fig. 22.27 | Getting a list of titles grouped by authors. (Part 1 of 2.)

```
            // display titles written by each author, grouped by author
            foreach ( var author in titlesByAuthor )
            {
               // display author's name
               outputTextBox.AppendText( "\r\n\t" + author.Name + ":" );

               // display titles written by that author
               foreach ( var title in author.Titles )
               {
                  outputTextBox.AppendText( "\r\n\t\t" + title );
               } // end inner foreach
            } // end outer foreach
         } // end method JoiningTableData_Load
      } // end class JoiningTableData
   } // end namespace JoinQueries
```

Fig. 22.27 | Getting a list of titles grouped by authors. (Part 2 of 2.)

a) Use the `BindingNavigator`'s controls to navigate through the contacts in the database

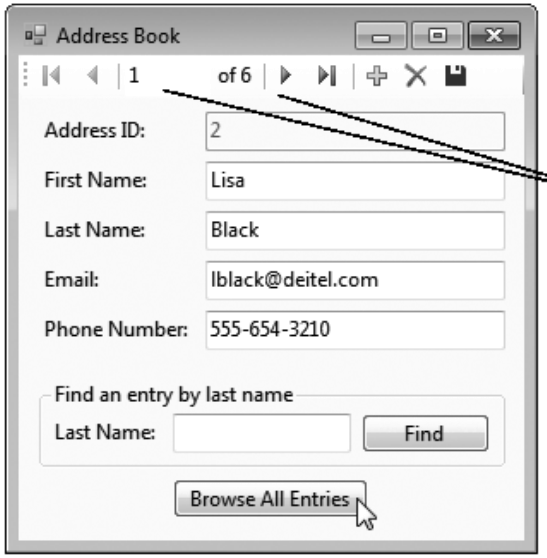
Fig. 22.32 | Manipulating an address book. (Part 1 of 3.)

b) Type a search `string` in the **Last Name:** `TextBox` then press **Find** to locate contacts whose last names begin with that `string`; only two names start with "Br" so the `BindingNavigator` indicates two matching records

Displaying the first of two matching contacts for the current search

Fig. 22.32 | Manipulating an address book. (Part 2 of 3.)

c) Click **Browse All Entries** to clear the search `string` and allow browsing of all contacts in the database



You can now browse through all six contacts

Fig. 22.32 | Manipulating an address book. (Part 3 of 3.)

```
// Fig. 22.33: Contact.cs
// Manipulating an address book.
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Validation;
using System.Linq;
using System.Windows.Forms;

namespace AddressBook
{
   public partial class Contacts : Form
   {
      public Contacts()
      {
         InitializeComponent();
      } // end constructor

      // Entity Framework DbContext
      private AddressExample.AddressBookEntities dbcontext = null;

```

Fig. 22.33 | Creating the `BooksDataContext` and defining method `RefreshContacts` for use in other methods. (Part 1 of 2.)

```
        // fill our addressBindingSource with all rows, ordered by name
        private void RefreshContacts()
        {
           // Dispose old DbContext, if any
           if ( dbcontext != null )
              dbcontext.Dispose();

           // create new DbContext so we can reorder records based on edits
           dbcontext = new AddressExample.AddressBookEntities();

           // use LINQ to order the Addresses table contents
           // by last name, then first name
           dbcontext.Addresses
              .OrderBy( entry => entry.LastName )
              .ThenBy( entry => entry.FirstName )
              .Load();

           // specify DataSource for addressBindingSource
           addressBindingSource.DataSource = dbcontext.Addresses.Local;
           addressBindingSource.MoveFirst(); // go to first result
           findTextBox.Clear(); // clear the Find TextBox
        } // end method RefreshContacts

```

Fig. 22.33 | Creating the `BooksDataContext` and defining method `RefreshContacts` for use in other methods. (Part 2 of 2.)

```
        // when the form loads, fill it with data from the database
        private void Contacts_Load( object sender, EventArgs e )
        {
           RefreshContacts(); // fill binding with data from database
        } // end method Contacts_Load

```

Fig. 22.34 | Calling `RefreshContacts` to fill the `TextBoxes` when the app loads.

```
        // Click event handler for the Save Button in the
        // BindingNavigator saves the changes made to the data
        private void addressBindingNavigatorSaveItem_Click(
           object sender, EventArgs e )
        {
           Validate(); // validate input fields
           addressBindingSource.EndEdit(); // complete current edit, if any

           // try to save changes
           try
           {
              dbcontext.SaveChanges(); // write changes to database file
           } // end try
           catch ( DbEntityValidationException )
           {
              MessageBox.Show( "Columns cannot be empty",
                 "Entity Validation Exception" );
           } // end catch

           RefreshContacts(); // change back to initial unfiltered data
        } // end method addressBindingNavigatorSaveItem_Click

```

Fig. 22.35 | Saving changes to the database when the user clicks Save Data.

```
        // use LINQ to create a data source that contains only people
        // with last names that start with the specified text
        private void findButton_Click( object sender, EventArgs e )
        {
           // use LINQ to filter contacts with last names that
           // start with findTextBox contents
           var lastNameQuery =
              from address in dbcontext.Addresses
              where address.LastName.StartsWith( findTextBox.Text )
              orderby address.LastName, address.FirstName
              select address;

           // display matching contacts
           addressBindingSource.DataSource = lastNameQuery.ToList();
           addressBindingSource.MoveFirst(); // go to first result

           // don't allow add/delete when contacts are filtered
           bindingNavigatorAddNewItem.Enabled = false;
           bindingNavigatorDeleteItem.Enabled = false;
        } // end method findButton_Click

```

Fig. 22.36 | Finding the contacts whose last names begin with a specified `String`.

```
        // reload addressBindingSource with all rows
        private void browseAllButton_Click( object sender, EventArgs e )
        {
           // allow add/delete when contacts are not filtered
           bindingNavigatorAddNewItem.Enabled = true;
           bindingNavigatorDeleteItem.Enabled = true;
           RefreshContacts(); // change back to initial unfiltered data
        } // end method browseButton_Click
     } // end class Contacts
  } // end namespace AddressBook
```

Fig. 22.37 | Allowing the user to browse all contacts.