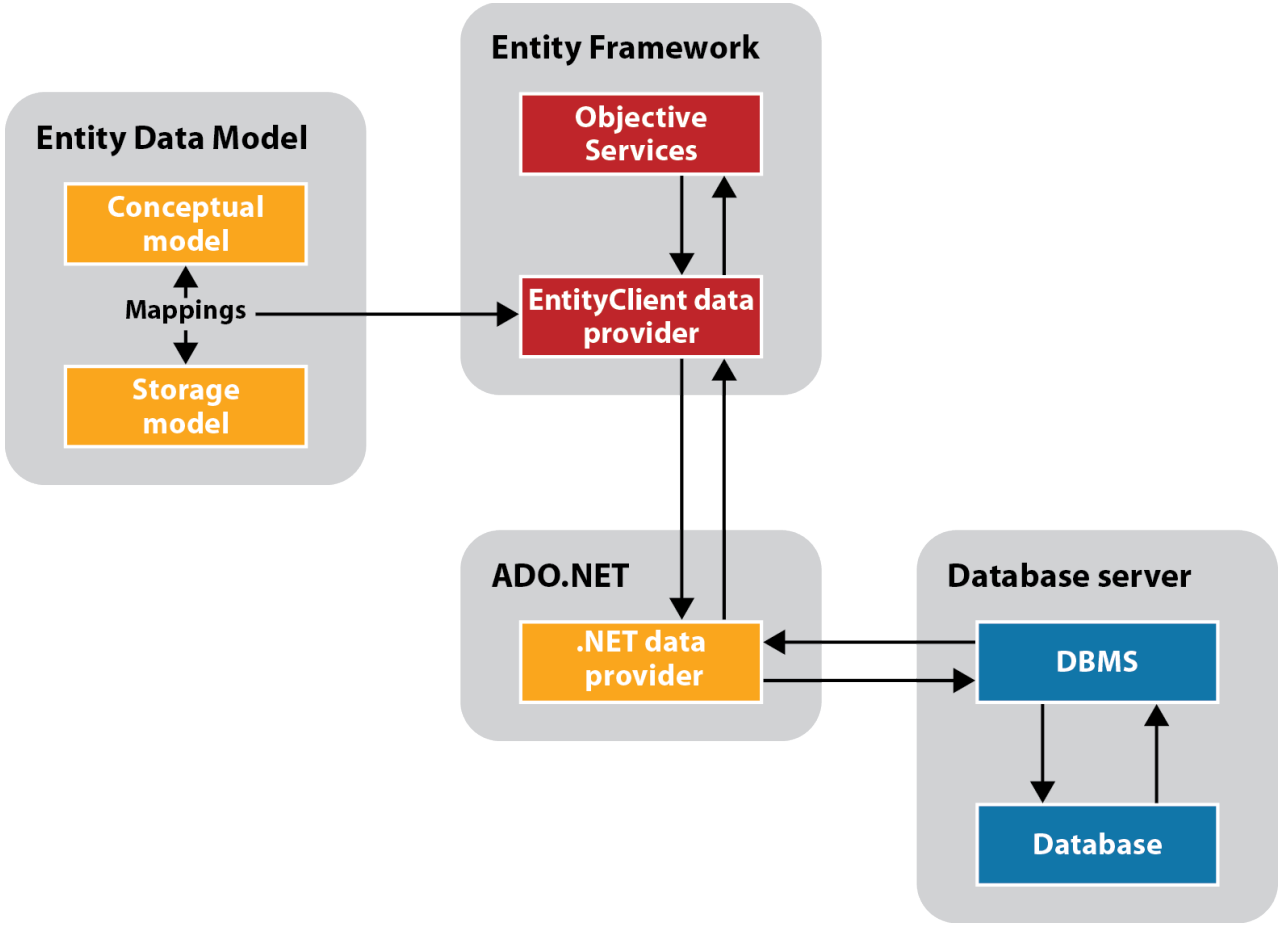
Chapter 24

# Must Know:

# How to use the Entity Framework

# And LINQ

# How the Entity Framework works



# Entity Framework concepts

- The *Entity Framework* provides a layer between the database used by an application and the objects used by an application. This layer provides an interface that allows the data in the database to be presented to the application as objects.
- To provide the interface between objects and database, the Entity Framework uses an *Entity Data Model* that defines a ***conceptual model*, a *storage model*, and *mappings*** between the two models.
- When you execute a query against a conceptual model, *Object Services* works with the *EntityClient data provider* and the Entity Data Model to translate the query into one that can be executed by the database. When the results are returned from the database, Object Services translates them back to the objects defined by the conceptual model.
- The Entity Framework also provides for tracking changes and for submitting those changes to the database.

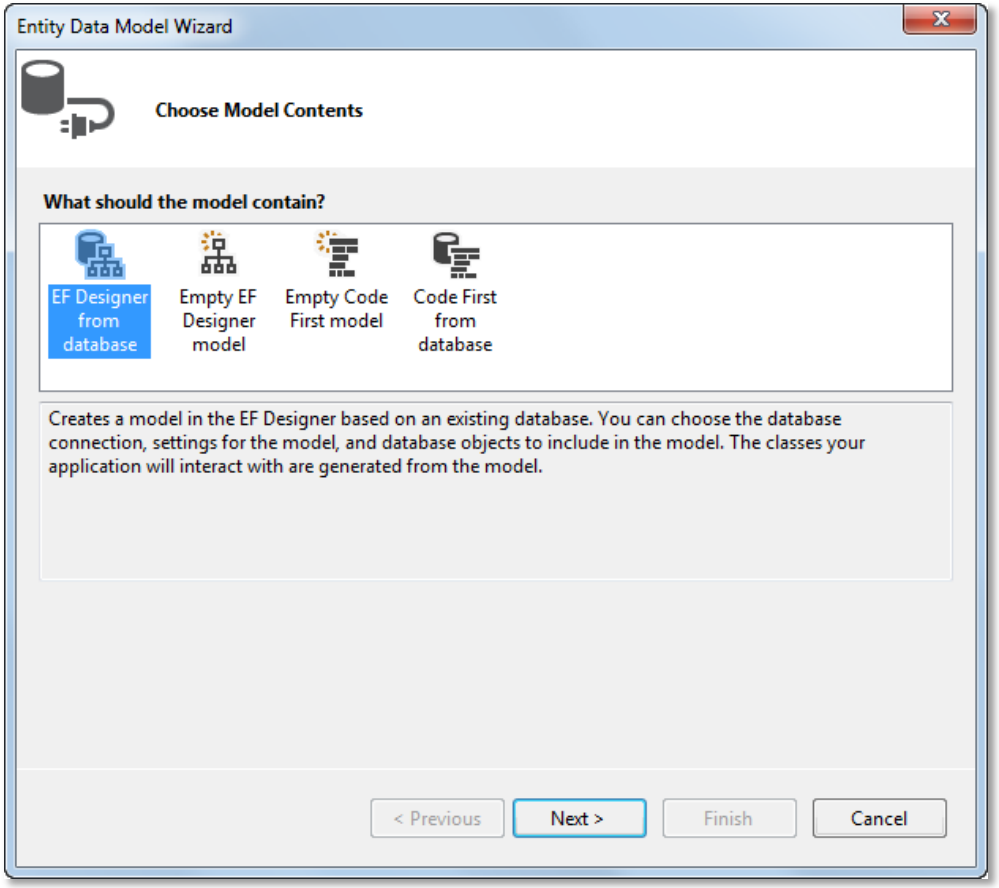
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/terminology

# Three ways to query a conceptual model

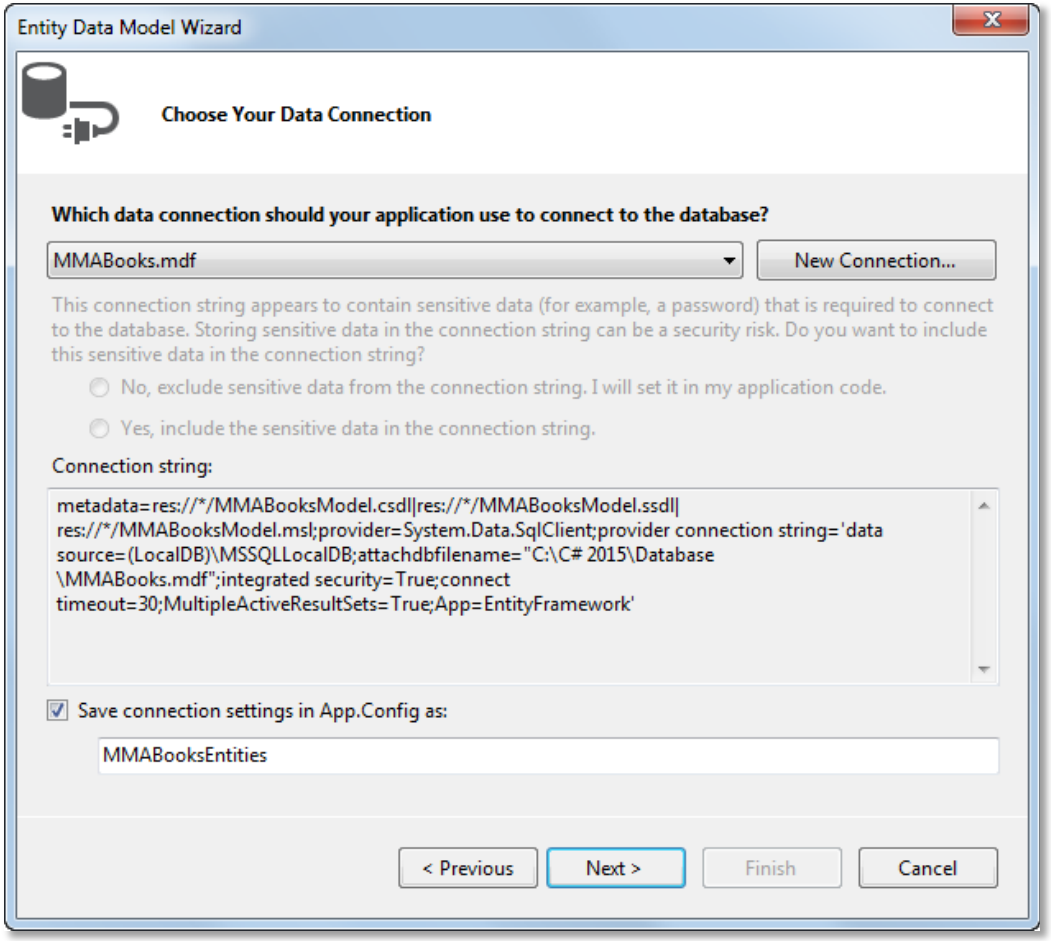
- LINQ to Entities
- Entity SQL
- Query builder methods

Linq to Entity is the easier way to build most queries and it's strongly typed

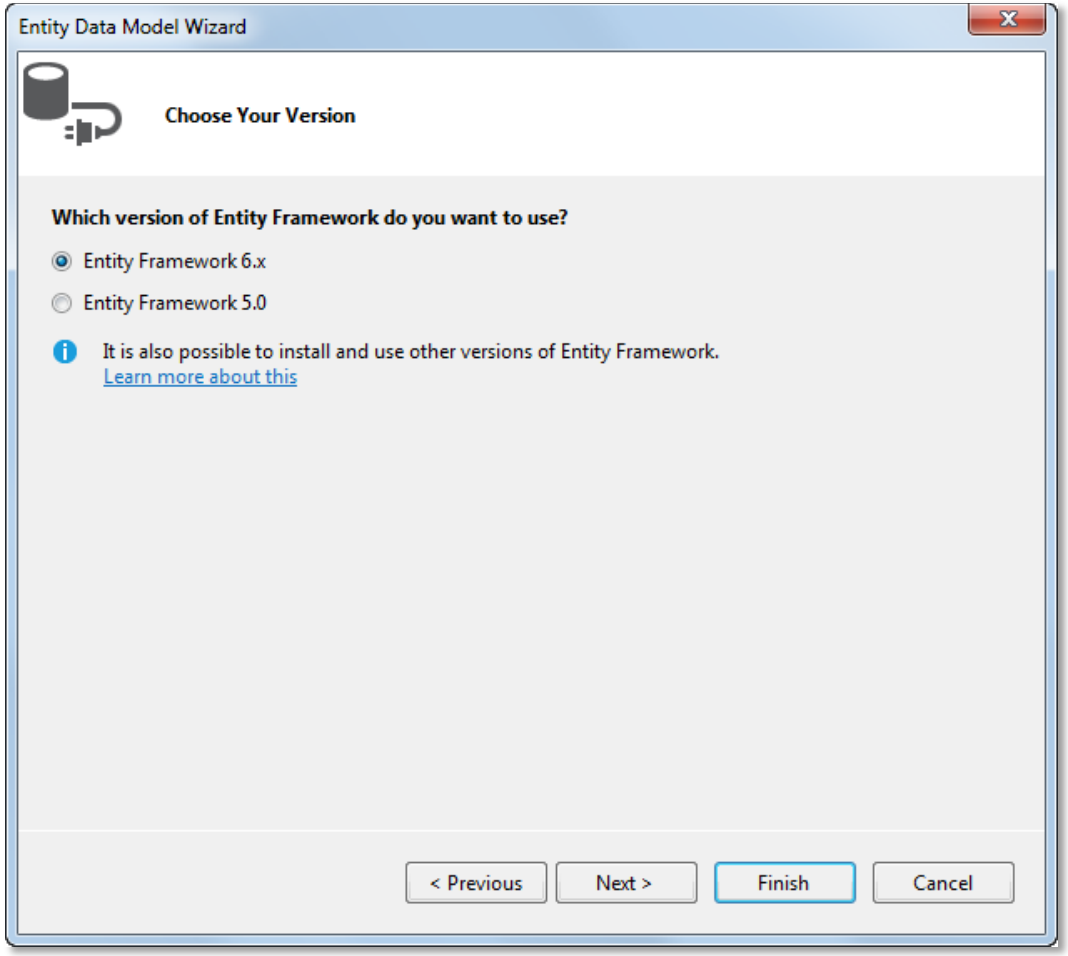
# The Entity Data Model Wizard: Step 1



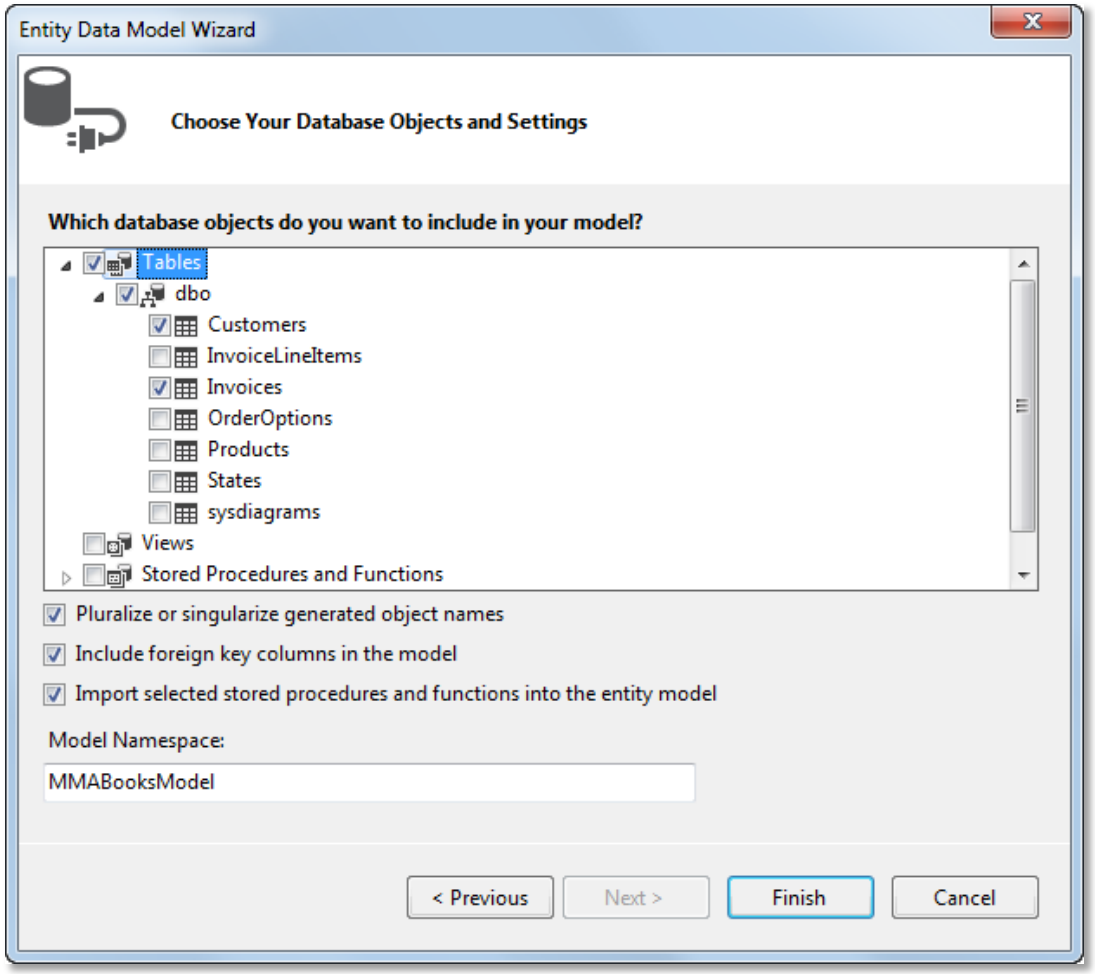
# The Entity Data Model Wizard: Step 2



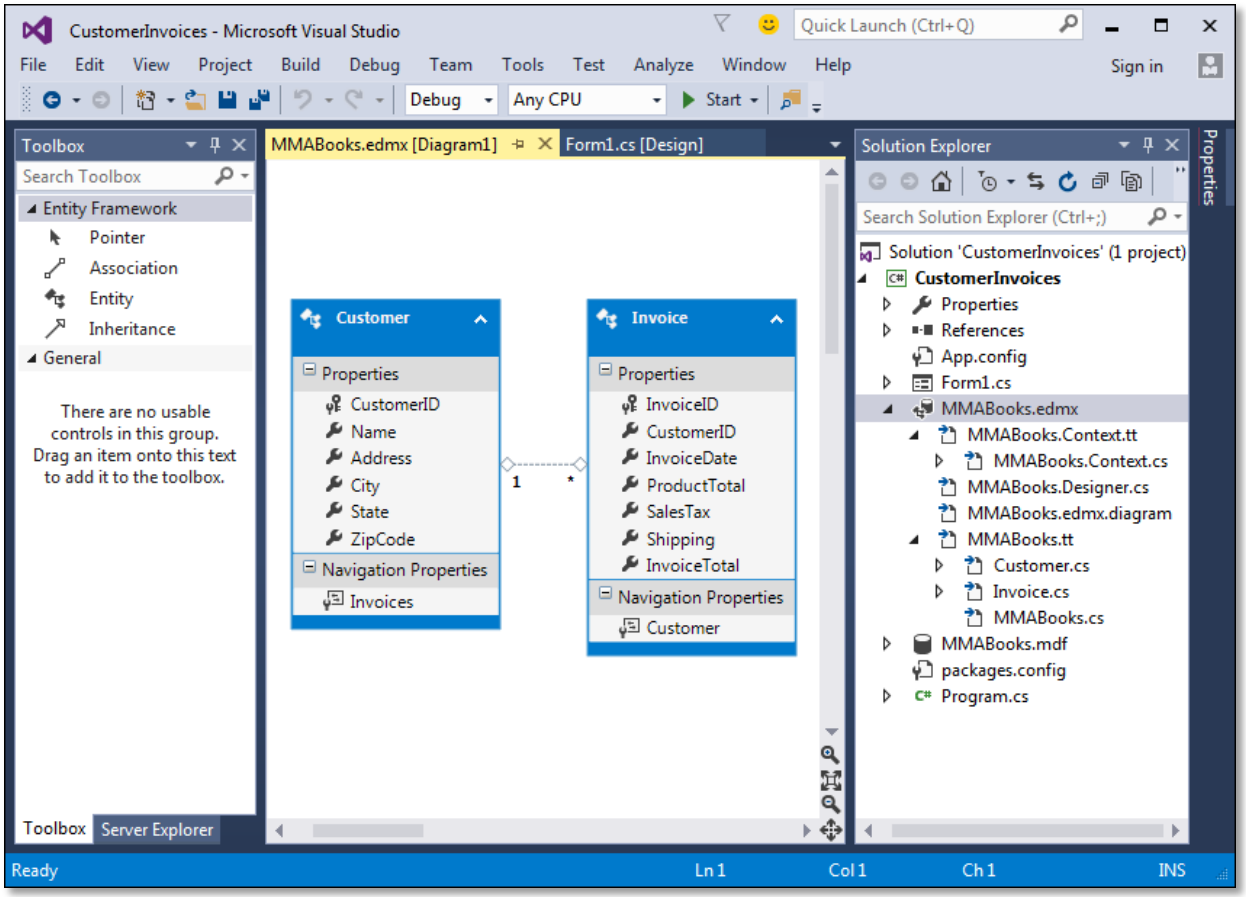
# The Entity Data Model Wizard: Step 3



# The Entity Data Model Wizard: Step 4



# The Entity Data Model in the Entity Data Model Designer



# The Entity Data Model Designer

CustomerInvoices - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test Analyze Window Help

Debug | Any CPU | Start

MMABooksModel.edmx [Diagram1]* | Form1.cs [Design]

Customer
- Properties
  - CustomerID
  - Name
  - Address
  - City
  - State
  - ZipCode
- Navigation Properties
  - Invoices

1 — *

Invoice
- Properties
  - InvoiceID
  - CustomerID
  - InvoiceDate
  - ProductTotal
  - SalesTax
  - Shipping
  - InvoiceTotal
- Navigation Properties
  - Customer

Model Browser

Type here to search

- MMABooksModel.edmx
  - Diagrams
    - Diagram1
  - MMABooksModel
    - Entity Types
    - Complex Types
    - Enum Types
    - Associations
    - Function Imports
    - EntityContainer: MMABooksEntities
  - MMABooksModel.Store
    - Tables / Views
    - Stored Procedures / Functions
    - Constraints

Solution Explorer | Model Browser

Mapping Details - Customer

| Column | Operator | Value / Property |
|---|---|---|
| Tables | | |
| Maps to Customers | | |
| <Add a Condition> | | |
| Column Mappings | | |
| CustomerID : int | ↔ | CustomerID : Int32 |
| Name : varchar | ↔ | Name : String |

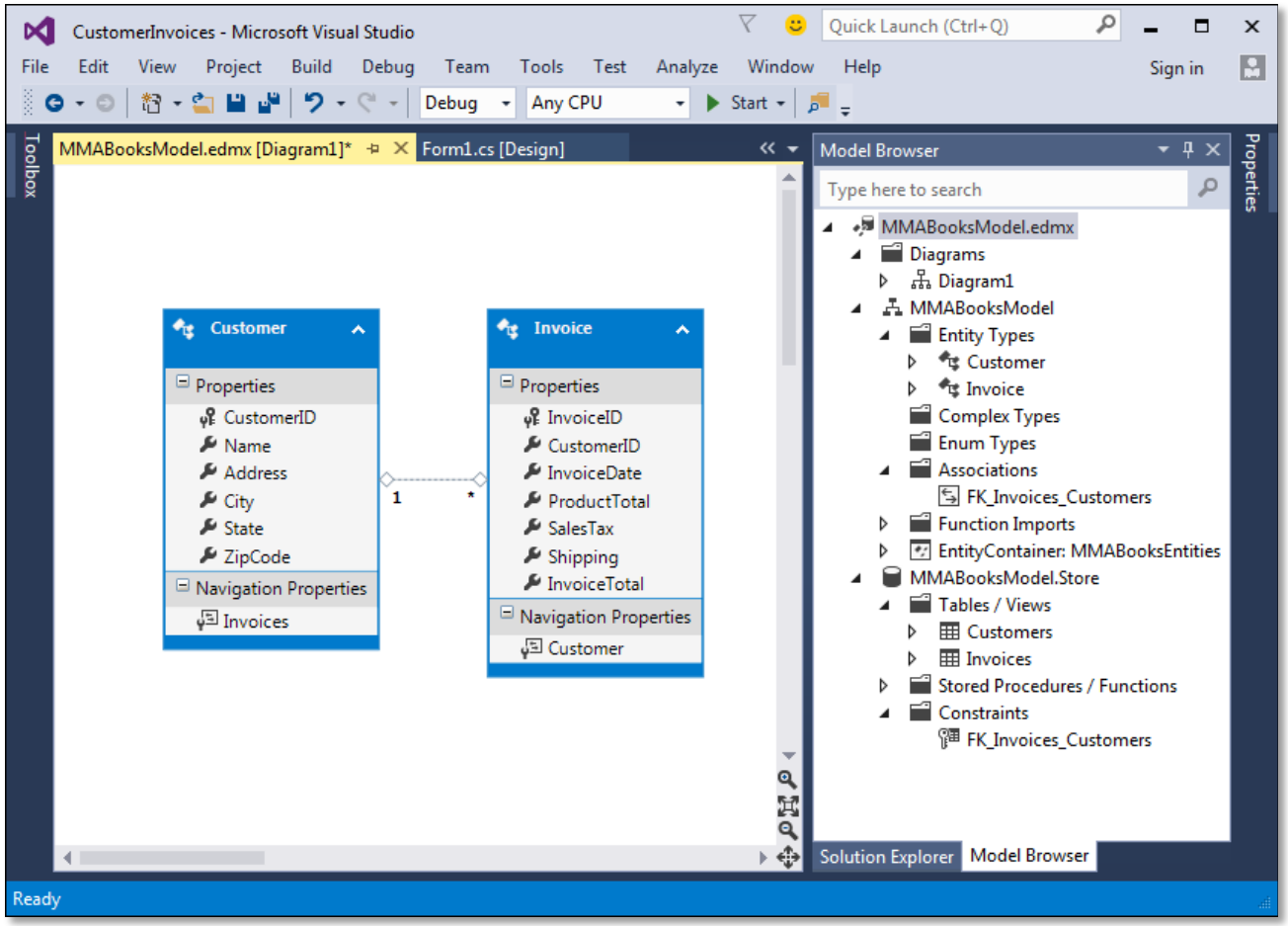
Properties

MMABooksModel.Customer EntityType

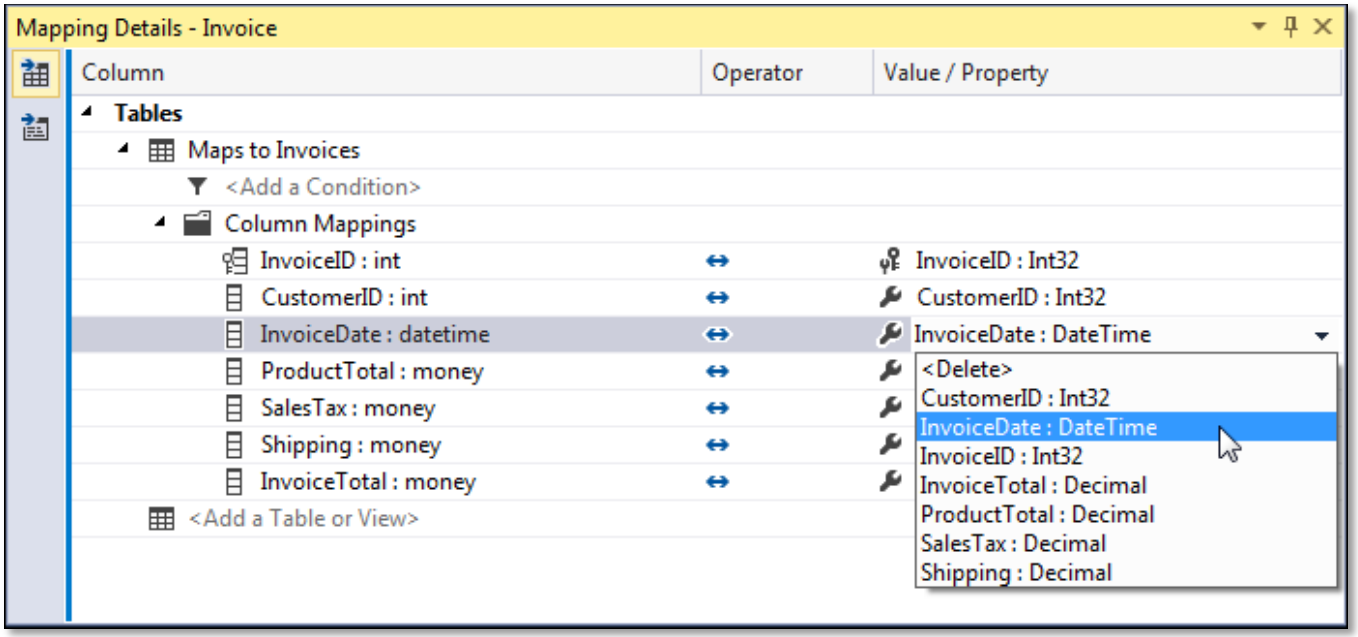
| | |
|---|---|
| Base Type | (None) |
| Documentation | |
| Entity Set Name | **Customers** |
| Name | **Customer** |

Ready

# The Model Browser window with some of its nodes expanded



# A Mapping Details window that displays the mappings for the Invoice entity



# A LINQ query that gets data from the Invoices table

## A statement that creates an instance of the object context

```
MMABooksEntities mmaBooks = new MMABooksEntities();
```

## A query expression that retrieves invoices over $200

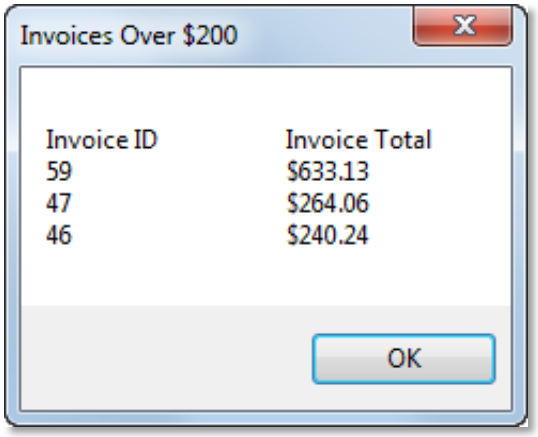
```
var highInvoices = from invoice in mmaBooks.Invoices
                   where invoice.InvoiceTotal > 200
                   orderby invoice.InvoiceTotal descending
                   select new { invoice.InvoiceID,
                                invoice.InvoiceTotal };
```

# A LINQ query that gets data from the Invoices table (cont.)

## Code that executes the query

```
string displayResult = "Invoice ID\t\tInvoice Total\n";
foreach (var invoice in highInvoices)
{
    displayResult += invoice.InvoiceID + "\t\t" +
        invoice.InvoiceTotal.ToString("c") + "\n";
}
MessageBox.Show(displayResult, "Invoices Over $200");
```

## The resulting dialog box



# A LINQ query that gets invoice data through the Customer object

```
var invoicesList = from customer in mmaBooks.Customers
                   from invoice in customer.Invoices
                   orderby customer.Name, invoice.InvoiceDate
                   select new { customer.Name, invoice.InvoiceID,
                                invoice.InvoiceTotal };
```

## A query expression that uses a navigation property to load related objects

```
var customerInvoices =
    (from customer in mmaBooks.Customers
     where customer.CustomerID == customerID
     select new { customer.Name, customer.Invoices }).Single();
```

## A query expression that uses the Include method to load related objects

```
var selectedCustomer =
    (from customer in mmaBooks.Customers.Include("Invoices")
     where customer.CustomerID == customerID
     select customer).Single();
```

## Code that explicitly loads the objects on the many side of a relationship

```
var selectedCustomer = (from customer in mmaBooks.Customers
                        where customer.CustomerID == customerID
                        select customer).Single();

if (! mmaBooks.Entry(selectedCustomer).Collection(
        "Invoices").IsLoaded)
    mmaBooks.Entry(selectedCustomer).Collection(
        "Invoices").Load();
```

## Code that explicitly loads the object on the one side of a relationship

```
var selectedInvoice = (from invoice in mmaBooks.Invoices
                       where invoice.InvoiceID == invoiceID
                       select invoice).Single();

if (! mmaBooks.Entry(selectedInvoice).Reference(
        "Customer").IsLoaded)
    mmaBooks.Entry(selectedInvoice).Reference("Customer").Load();
```

## Code that retrieves a customer row

```
var selectedCustomer =
    (from customer in mmaBooks.Customers
     where customer.CustomerID == CustomerID
     select customer).Single();
```

## Code that modifies the data in the customer row

```
selectedCustomer.Name = txtName.Text;
selectedCustomer.City = txtCity.Text;
```

## A statement that saves the changes to the database

```
mmaBooks.SaveChanges();
```

# Code that assigns an invoice to a different customer

```
int invoiceID = Convert.ToInt32(txtInvoiceID.Text);
var selectedInvoice =
    (from invoice in mmaBooks.Invoices
     where invoice.InvoiceID == invoiceID
     select invoice).Single();
selectedInvoice.Customer = selectedCustomer;
```

## Code that retrieves an invoice row and its related line item rows

```
var selectedInvoice =
    (from invoice in mmaBooks.Invoices.Include(
        "InvoiceLineItems")
     where invoice.InvoiceID == Convert.ToInt32(
         txtInvoiceID.Text)
     select invoice).Single();
```

## A statement that marks the Invoice object for deletion

```
mmaBooks.Invoices.Remove(selectedInvoice);
```

## A statement that deletes the invoice and line items from the database

```
mmaBooks.SaveChanges();
```

## Code that creates a new Invoice object

```
Invoice newInvoice = new Invoice {
    CustomerID = 14,
    InvoiceDate = new DateTime(2016, 01, 04),
    ProductTotal = 156.00m,
    SalesTax = 11.70m,
    Shipping = 6.25m,
    InvoiceTotal = 173.95m };
```

## Code that adds the object to the Invoices collection and updates the database

```
mmabooks.Invoices.Add(newInvoice);
mmabooks.SaveChanges();
```

## Code that creates a new InvoiceLineItem object

```
InvoiceLineItem newLineItem = new InvoiceLineItem {
    InvoiceID = newInvoice.InvoiceID,
    ProductCode = "A46V",
    UnitPrice = 57.50m,
    Quantity = 1,
    ItemTotal = 57.50m };
```

## Code that adds the object to the InvoiceLineItems collection and updates the database

```
mmabooks.InvoiceLineItems.Add(newLineItem);
mmabooks.SaveChanges();
```

## Another way to add the object to the InvoiceLineItems collection

```
newInvoice.InvoiceLineItems.Add(newLineItem);
```

# A try-catch statement that uses store wins for concurrency exceptions

```
try
{
    mmaBooks.SaveChanges();
}
catch (DbUpdateConcurrencyException ex)
{
    ex.Entries.Single().Reload();
    ...
}
```

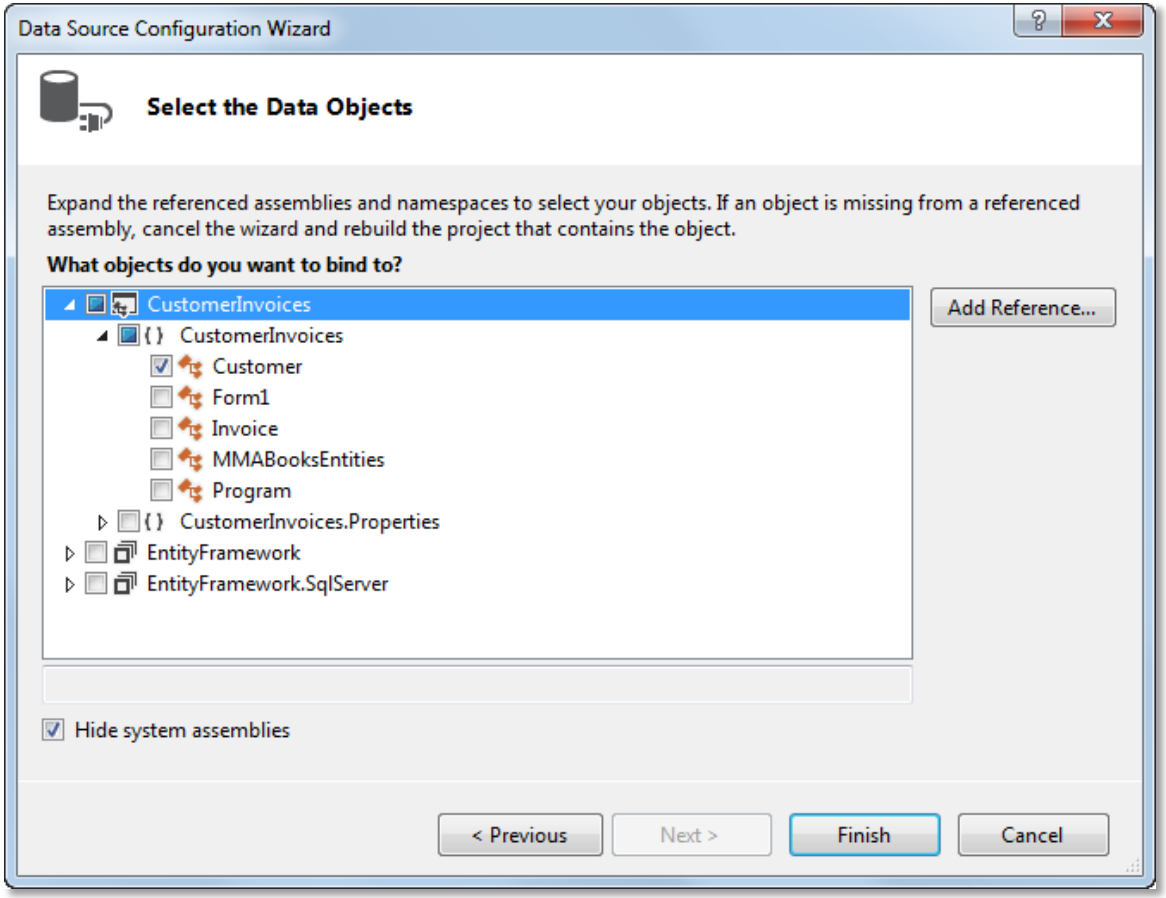
## Code in a catch block that uses client wins for concurrency exceptions

```
catch (DbUpdateConcurrencyException ex)
{
    var entry = ex.Entries.Single();
    entry.OriginalValues.SetValues(
        entry.GetDatabaseValues());
    ...
}
```

## Checking if a currency exception occurred due to the row being deleted

```
if (mmaBooks.Entry(customer).State ==
    EntityState.Detached) ...
```

# The Data Source Configuration Wizard: Step 2



## Code that binds a combo box to an entity collection

```
cboCustomers.DataSource = mmaBooks.Customers.ToList();
cboCustomers.DisplayMember = "Name";
cboCustomers.ValueMember = "CustomerID";
```

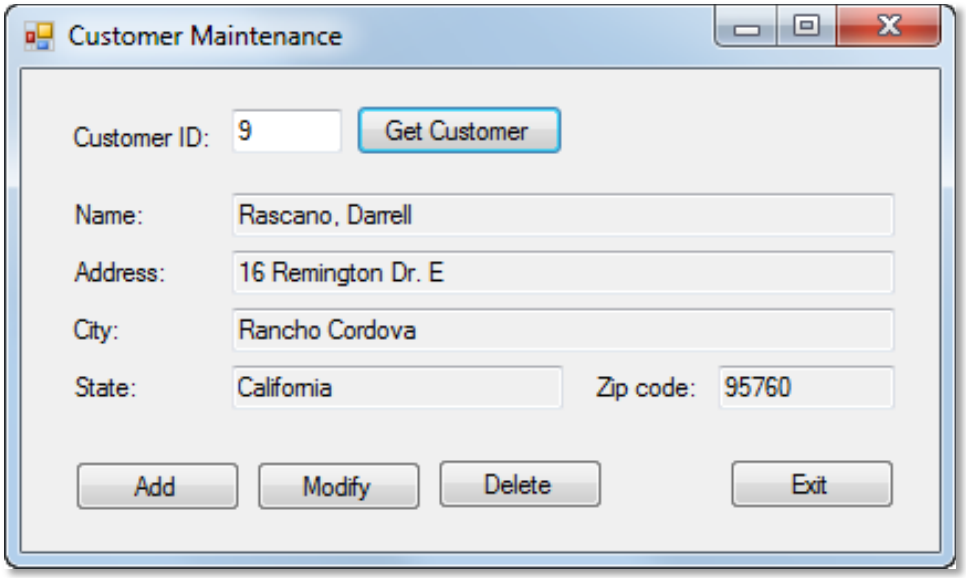
## Code that binds a combo box to the results of a query

```
var customers =
    from customer in mmaBooks.Customers
    orderby customer.Name
    select new { customer.CustomerID, customer.Name };

cboCustomers.DataSource = customers.ToList();
cboCustomers.DisplayMember = "Name";
cboCuatomers.ValueMember = "CustomerID";
```

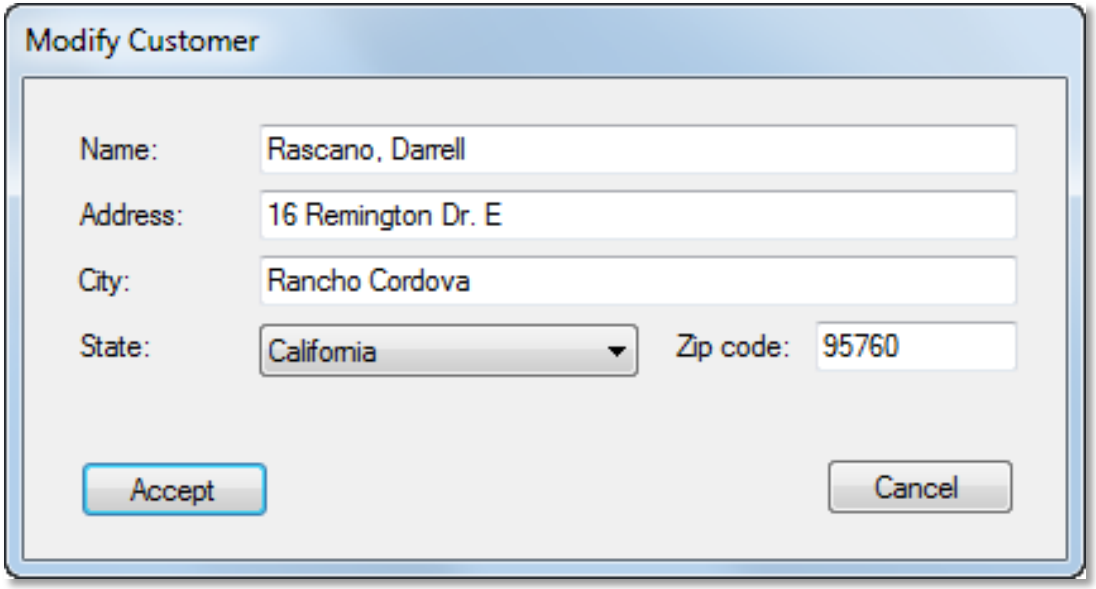
## A statement that binds a combo box using its binding source

```
customerBindingSource.DataSource = customers.ToList();
```

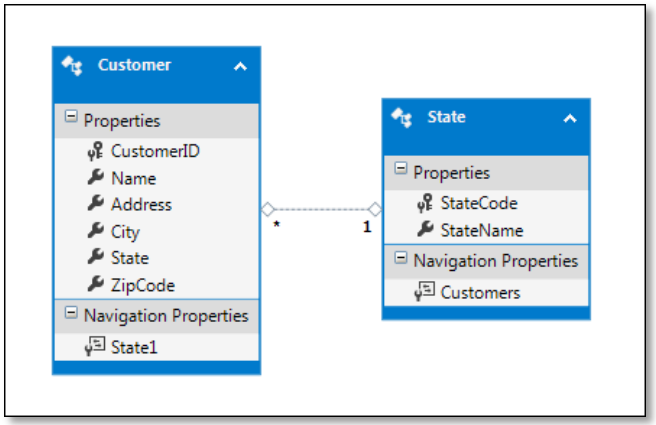
# The Customer Maintenance Form



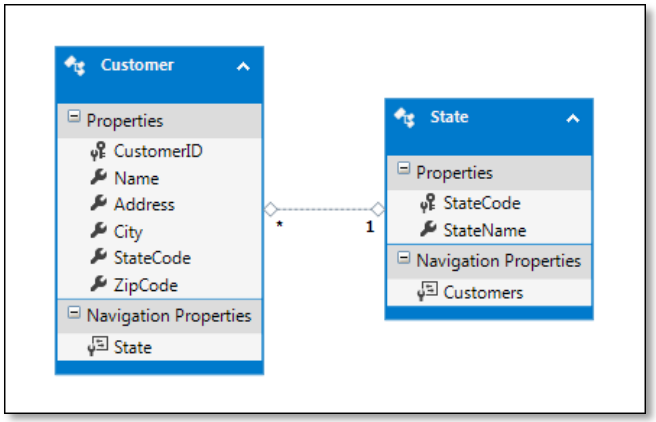
# The Add/Modify Customer form



# The default Entity Data Model




# The modified Entity Data Model



# The code for the MMABooksEntity class

```
public static class MMABooksEntity
{
    public static MMABooksEntities mmaBooks =
        new MMABooksEntities();
}
```

# The Customer Maintenance form

```
public partial class frmCustomerMaintenance : Form
{
    public frmCustomerMaintenance()
    {
        InitializeComponent();
    }

    private Customer selectedCustomer;

    private void btnGetCustomer_Click(object sender, EventArgs e)
    {
        if (Validator.IsPresent(txtCustomerID) &&
            Validator.IsInt32(txtCustomerID))
        {
            int customerID = Convert.ToInt32(txtCustomerID.Text);
            this.GetCustomer(customerID);
        }
    }

    private void GetCustomer(int CustomerID)
    {
        try
        {
            selectedCustomer =
                (from customer in MMABooksEntity.mmaBooks.Customers
                 where customer.CustomerID == CustomerID
                 select customer).SingleOrDefault();
```

# The Customer Maintenance form (cont.)

```
            if (selectedCustomer == null)
            {
                MessageBox.Show("No customer found with this ID. " +
                    "Please try again.", "Customer Not Found");
                this.ClearControls();
                txtCustomerID.Focus();
            }
            else
            {
                if (!MMABooksEntity.mmaBooks.Entry(
                       selectedCustomer).Reference("State").IsLoaded)
                    MMABooksEntity.mmaBooks.Entry(
                        selectedCustomer).Reference("State").Load();
                this.DisplayCustomer();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, ex.GetType().ToString());
        }
    }
```

# The Customer Maintenance form (cont.)

```
private void DisplayCustomer()
{
    txtName.Text = selectedCustomer.Name;
    txtAddress.Text = selectedCustomer.Address;
    txtCity.Text = selectedCustomer.City;
    txtState.Text = selectedCustomer.State.StateName;
    txtZipCode.Text = selectedCustomer.ZipCode;
    btnModify.Enabled = true;
    btnDelete.Enabled = true;
}

private void ClearControls()
{
    txtName.Text = "";
    txtAddress.Text = "";
    txtCity.Text = "";
    txtState.Text = "";
    txtZipCode.Text = "";
    btnModify.Enabled = false;
    btnDelete.Enabled = false;
}

private void btnAdd_Click(object sender, EventArgs e)
{
    frmAddModifyCustomer addModifyCustomerForm =
        new frmAddModifyCustomer();
    addModifyCustomerForm.addCustomer = true;
    DialogResult result = addModifyCustomerForm.ShowDialog();
```

# The Customer Maintenance form (cont.)

```
        if (result == DialogResult.OK)
        {
            selectedCustomer = addModifyCustomerForm.customer;
            txtCustomerID.Text = selectedCustomer.CustomerID.ToString();
            this.DisplayCustomer();
        }
    }

    private void btnModify_Click(object sender, EventArgs e)
    {
        frmAddModifyCustomer addModifyCustomerForm =
            new frmAddModifyCustomer();
        addModifyCustomerForm.addCustomer = false;
        addModifyCustomerForm.customer = selectedCustomer;
        DialogResult result = addModifyCustomerForm.ShowDialog();
        if (result == DialogResult.OK || result == DialogResult.Retry)
        {
            selectedCustomer = addModifyCustomerForm.customer;
            this.DisplayCustomer();
        }
        else
        {
            txtCustomerID.Text = "";
            this.ClearControls();
        }
    }
```

# The Customer Maintenance form (cont.)

```
private void btnDelete_Click(object sender, EventArgs e)
{
    DialogResult result =
        MessageBox.Show("Delete " + selectedCustomer.Name + "?",
        "Confirm Delete", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question);
    if (result == DialogResult.Yes)
    {
        try
        {
            MMABooksEntity.mmaBooks.Customers.Remove(selectedCustomer);
            MMABooksEntity.mmaBooks.SaveChanges();
            txtCustomerID.Text = "";
            this.ClearControls();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            ex.Entries.Single().Reload();
            if (MMABooksEntity.mmaBooks.Entry(selectedCustomer).State ==
                    EntityState.Detached)
            {
                MessageBox.Show("Another user has deleted " +
                    "that customer.", "Concurrency Error");
                txtCustomerID.Text = "";
                this.ClearControls();
            }
```

# The Customer Maintenance form (cont.)

```
                else
                {
                    MessageBox.Show("Another user has updated " +
                        "that customer.", "Concurrency Error");
                    this.DisplayCustomer();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, ex.GetType().ToString());
            }
        }
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

# The Add/Modify Customer form

```
public partial class frmAddModifyVendor : Form
{
    public bool addCustomer;
    public Customer customer;

    private void frmAddModifyCustomer_Load(object sender, EventArgs e)
    {
        this.LoadComboBox();
        if (addCustomer)
        {
            this.Text = "Add Customer";
            cboStates.SelectedIndex = -1;
        }
        else
        {
            this.Text = "Modify Customer";
            this.DisplayCustomerData();
        }
    }
```

# The Add/Modify Customer form (cont.)

```
private void LoadComboBox()
{
    try
    {
        var states = (from state in MMABooksEntity.mmaBooks.States
                      orderby state.StateName
                      select state).ToList();

        cboStates.DataSource = states;
        cboStates.DisplayMember = "StateName";
        cboStates.ValueMember = "StateCode";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, ex.GetType().ToString());
    }
}

private void DisplayCustomerData()
{
    txtName.Text = customer.Name;
    txtAddress.Text = customer.Address;
    txtCity.Text = customer.City;
    cboStates.SelectedValue = customer.StateCode;
    txtZipCode.Text = customer.ZipCode;
}
```

# The Add/Modify Customer form (cont.)

```
private void btnAccept_Click(object sender, EventArgs e)
{
    if (IsValidData())
    {
        if (addCustomer)
        {
            customer = new Customer();
            this.PutCustomerData(customer);
            MMABooksEntity.mmaBooks.Customers.Add(customer);
            try
            {
                MMABooksEntity.mmaBooks.SaveChanges();
                this.DialogResult = DialogResult.OK;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, ex.GetType().ToString());
            }
        }
        else
        {
            this.PutCustomerData(customer);
            try
            {
                MMABooksEntity.mmaBooks.SaveChanges();
                this.DialogResult = DialogResult.OK;
            }
```

# The Add/Modify Customer form (cont.)

```
            catch (DbUpdateConcurrencyException ex)
            {
                ex.Entries.Single().Reload();
                if (MMABooksEntity.mmaBooks.Entry(customer).State ==
                        EntityState.Detached)
                {
                    MessageBox.Show("Another user has deleted " +
                        "that customer.", "Concurrency Error");
                    this.DialogResult = DialogResult.Abort;
                }
                else
                {
                    MessageBox.Show("Another user has updated " +
                        "that customer.", "Concurrency Error");
                    this.DialogResult = DialogResult.Retry;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, ex.GetType().ToString());
            }
        }
    }
}
```

# The Add/Modify Customer form (cont.)

```
    private bool IsValidData()
    {
         return Validator.IsPresent(txtName) &&
                Validator.IsPresent(txtAddress) &&
                Validator.IsPresent(txtCity) &&
                Validator.IsPresent(cboStates) &&
                Validator.IsPresent(txtZipCode) &&
                Validator.IsInt32(txtZipCode);
    }

    private void PutCustomerData(Customer customer)
    {
        customer.Name = txtName.Text;
        customer.Address = txtAddress.Text;
        customer.City = txtCity.Text;
        customer.StateCode = cboStates.SelectedValue.ToString();
        customer.ZipCode = txtZipCode.Text;
    }
}
```

# LINQ Introduction

# Some of the C# clauses for working with LINQ

```
from
where
orderby
select
join
```

# Features of LINQ

- Query language is integrated with C#
- Provides IntelliSense, compile-time syntax checking, and debugging support
- Same basic syntax for each type of query
- Provides designer tools that create *object-relational mappings*

# The three stages of a query operation

1. Get the data source. If the data source is an array, for example, you must declare the array and then assign values to its elements.
2. Define the query expression.
3. Execute the query to return the results.

# A LINQ query that retrieves data from an array

## Code that defines the array

```
int[] numbers = new int[6];
for (int i = 0; i < numbers.Length; i++)
    numbers[i] = i;
```

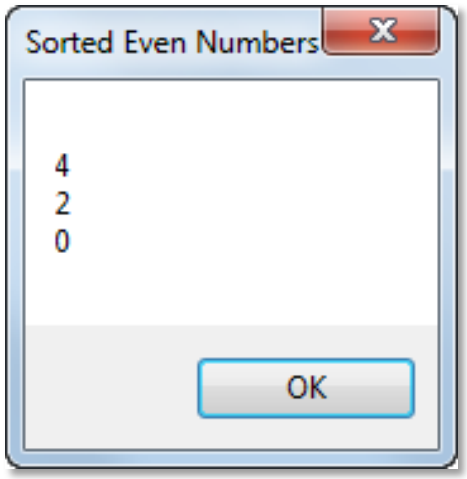
## A statement that defines the query expression

```
var numberList = from number in numbers
                 where number % 2 == 0
                 orderby number descending
                 select number;
```

## Code that executes the query

```
string numberDisplay = "";
foreach (var number in numberList)
    numberDisplay += number + "\t\t\n";
MessageBox.Show(numberDisplay, "Sorted Even Numbers");
```

# The resulting dialog box



## The syntax of the from clause

```
from [type] elementName in collectionName
```

# An example that uses an array of decimals

### A statement that gets the data source

```
decimal[] salesTotals =
    new decimal[4] {1286.45m, 2433.49m, 2893.85m,
        2094.53m};
```

### A statement that defines the query expression

```
var salesList = from sales in salesTotals
                select sales;
```

### Code that executes the query

```
decimal sum = 0;
foreach (var sales in salesList)
    sum += sales;
```

# An example that uses a generic list of invoices as the data source

## The Invoice class

```
public class Invoice
{
    public int InvoiceID { get; set; }
    public int CustomerID { get; set; }
    public DateTime InvoiceDate { get; set; }
    public decimal ProductTotal { get; set; }
    public decimal SalesTax { get; set; }
    public decimal Shipping { get; set; }
    public decimal InvoiceTotal { get; set; }
}
```

## A statement that gets the data source

```
List<Invoice> invoiceList = InvoiceDB.GetInvoices();
```

# An example that uses a generic list of invoices as the data source (cont.)

## A statement that defines the query expression

```
var invoices = from invoice in invoiceList
               select invoice;
```

## Code that executes the query

```
decimal sum = 0;
foreach (var invoice in invoices)
    sum += invoice.InvoiceTotal;
```

## The syntax of the where clause

```
where condition
```

# An example that filters the salesTotals array

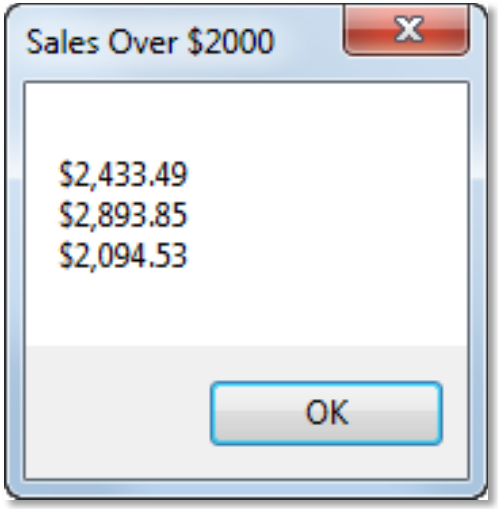
### A query expression that returns only sales greater than $2000

```
var salesList = from sales in salesTotals
                where sales > 2000
                select sales;
```

### Code that executes the query

```
string salesDisplay = "";
foreach (var sales in salesList)
    salesDisplay += sales.ToString("c") + "\t\t\n";
MessageBox.Show(salesDisplay, "Sales Over $2000");
```

# The resulting dialog box



# An example that filters the generic list of invoices

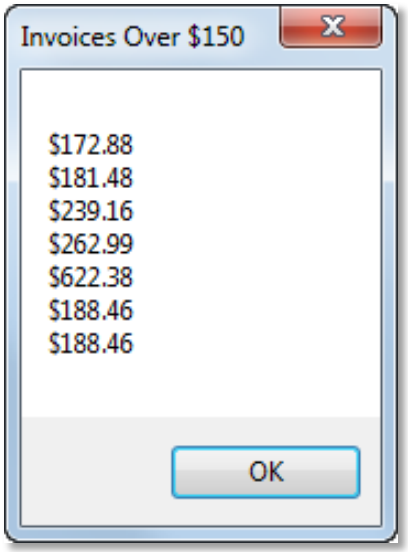
## A query expression that returns invoices with totals over $150

```
var invoices = from invoice in invoiceList
               where invoice.InvoiceTotal > 150
               select invoice;
```

## Code that executes the query

```
string invoiceDisplay = "";
foreach (var invoice in invoices)
    invoiceDisplay +=
        invoice.InvoiceTotal.ToString("c") + "\t\t\n";
MessageBox.Show(invoiceDisplay, "Invoices Over $150");
```

# The resulting dialog box



## The syntax of the orderby clause

```
orderby expression1 [ascending|descending]
      [, expression2 [ascending|descending]]...
```

## An example that sorts the salesTotals array

### A query expression that sorts the sales in ascending sequence

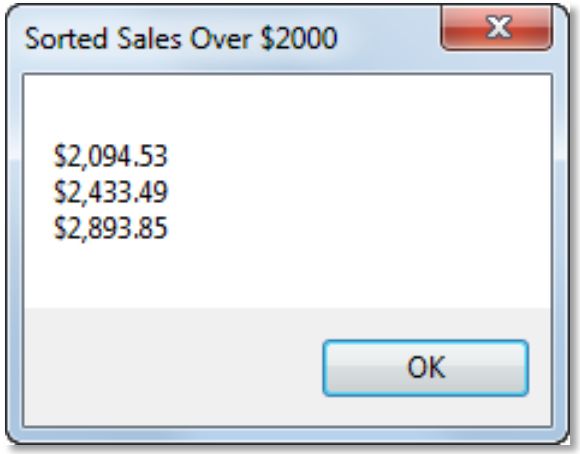
```
var salesList = from sales in salesTotals
                where sales > 2000
                orderby sales
                select sales;
```

### Code that executes the query

```
string salesDisplay = "";
foreach (var sales in salesList)
    salesDisplay += sales.ToString("c") + "\t\t\t\n";
MessageBox.Show(salesDisplay, "Sorted Sales Over $2000");
```

How about orderby multiple fields? Try it in VS.

# The resulting dialog box



# An example that sorts the generic list of invoices

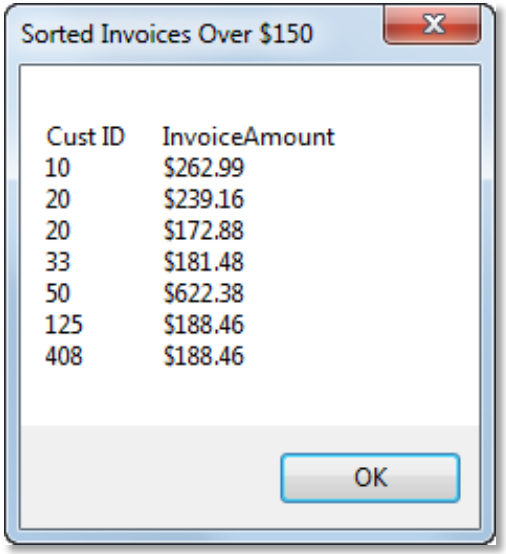
## A query expression that sorts the invoices by customer ID and invoice total

```
var invoices = from invoice in invoiceList
               where invoice.InvoiceTotal > 150
               orderby invoice.CustomerID,
                       invoice.InvoiceTotal descending
               select invoice;
```

## Code that executes the query

```
string invoiceDisplay = "Cust ID\tInvoice amount\n";
foreach (var invoice in invoices)
    invoiceDisplay += invoice.CustomerID + "\t"
                    + invoice.InvoiceTotal.ToString("c")
                    + "\n";
MessageBox.Show(invoiceDisplay,
    "Sorted Invoices Over $150");
```

# The resulting dialog box



# Two ways to code the select clause

```
select columnExpression

select new [type] { [PropertyName1 =] columnExpression1
                  [, [PropertyName2 =] columnExpression2]... }
```

# An example that selects key values from a sorted list

## The employee sales sorted list

```
SortedList<string, decimal> employeeSales =
    new SortedList<string, decimal>
    { ["Anderson"] = 1286.45m, ["Menendez"] = 2433.49m,
      ["Thompson"] = 2893.85m, ["Wilkinson"] = 2094.53m };
```

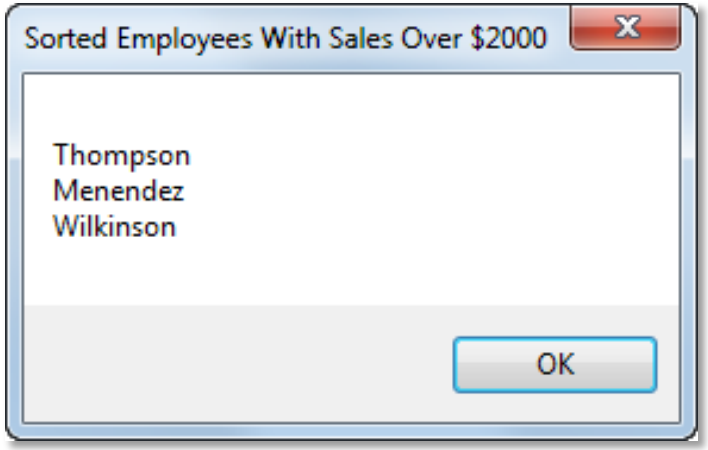
## A query expression that selects the employee names

```
var employeeList = from sales in employeeSales
                   where sales.Value > 2000
                   orderby sales.Value descending
                   select sales.Key;
```

## Code that executes the query

```
string employeeDisplay = "";
foreach (var employee in employeeList)
    employeeDisplay += employee + "\t\t\t\n";
MessageBox.Show(employeeDisplay,
    "Sorted Employees With Sales Over $2000");
```

# The resulting dialog box



# A query expression that creates an anonymous type from the list of invoices

```
var invoices = from invoice in invoiceList
               where invoice.InvoiceTotal > 150
               orderby invoice.CustomerID,
                       invoice.InvoiceTotal descending
               select new { invoice.CustomerID,
                            invoice.InvoiceTotal };
```

## The basic syntax of the join clause

```
join elementName in collectionName
    on keyName1 equals keyName2
```

## An example that joins data from two generic lists

### The Customer class

```
public class Customer
{
    public int CustomerID { get; set; }
    public string Name { get; set; }
}
```

### Code that gets the two data sources

```
List<Invoice> invoiceList = InvoiceDB.GetInvoices();
List<Customer> customerList = CustomerDB.GetCustomers();
```

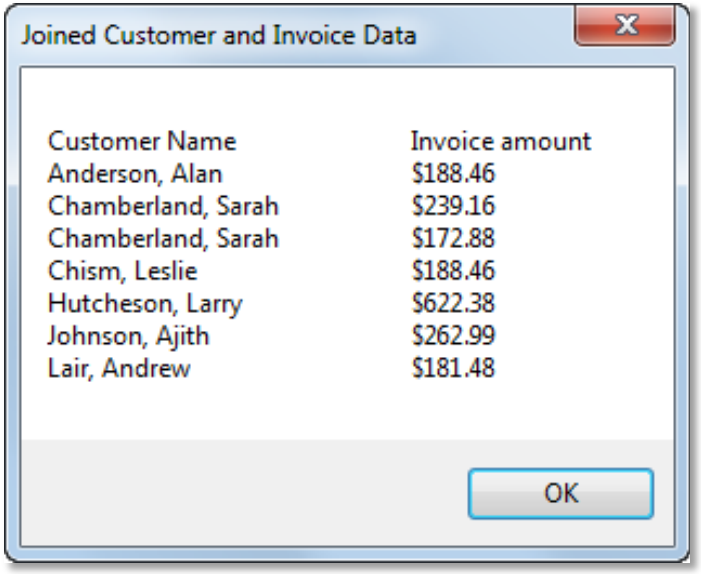
# A query expression that joins data from the two data sources

```
var invoices = from invoice in invoiceList
               join customer in customerList
               on invoice.CustomerID equals customer.CustomerID
               where invoice.InvoiceTotal > 150
               orderby customer.Name,
                       invoice.InvoiceTotal descending
               select new { customer.Name,
                            invoice.InvoiceTotal };
```

# Code that executes the query

```
string invoiceDisplay = "Customer Name\t\tInvoice amount\n";
foreach (var invoice in invoices)
{
    invoiceDisplay += invoice.Name + "\t\t";
    invoiceDisplay += invoice.InvoiceTotal.ToString("c") + "\n";
}
MessageBox.Show(invoiceDisplay,
     "Joined Customer and Invoice Data");
```

# The resulting dialog box



# An extension method that extends the String data type

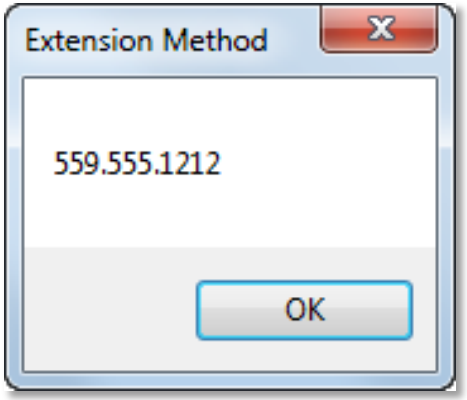
## A class with an extension method that formats a phone number

```
public static class StringExtensions
{
    public static string FormattedPhoneNumber(this string phone,
        string separator)
    {
        return phone.Substring(0, 3) + separator
            + phone.Substring(3, 3) + separator
            + phone.Substring(6, 4);
    }
}
```

## Code that uses the extension method

```
string phoneNumber = "5595551212";
string formattedPhoneNumber =
phoneNumber.FormattedPhoneNumber(".");
MessageBox.Show(formattedPhoneNumber + "\t", "Extension Method");
```

# The resulting dialog box



# Extension methods used to implement common C# clauses for LINQ

| Clause | Method |
|---|---|
| `where` | `Where` |
| `orderby` | `OrderBy, OrderByDescending, ThenBy, ThenByDescending` |
| `select` | `Select` |
| `join` | `Join` |

**Extension methods** enable you to add **methods** to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. An **extension method** is a special kind of static **method**, but they are called as if they were instance**methods** on the extended type.

## The basic syntax of a lambda expression

```
[(]parameterList[)] => expression
```

# A lambda expression that tests a condition

### A statement that declares the delegate type at the class level

```
delegate bool compareDel(decimal total);
```

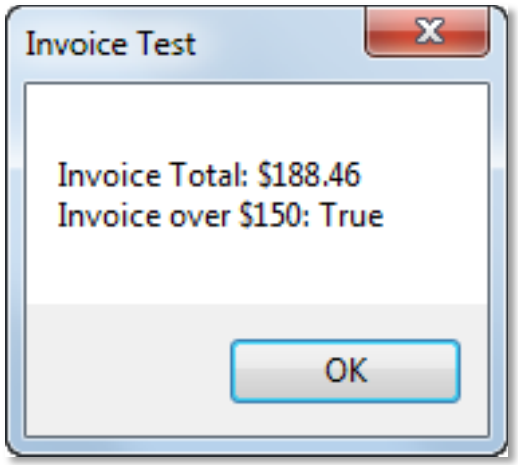
### A statement that defines the lambda expression and assigns it to a variable created from the delegate type

```
compareDel invoiceOver150 = total => total > 150;
```

### Code that executes the lambda expression

```
decimal invoiceTotal = 188.46m;
string invoiceMessage = "";
invoiceMessage += "Invoice Total: " +
                  invoiceTotal.ToString("c") +
                  "\n" + "Invoice over $150: " +
                  invoiceOver150(invoiceTotal);
MessageBox.Show(invoiceMessage, "Invoice Test");
```

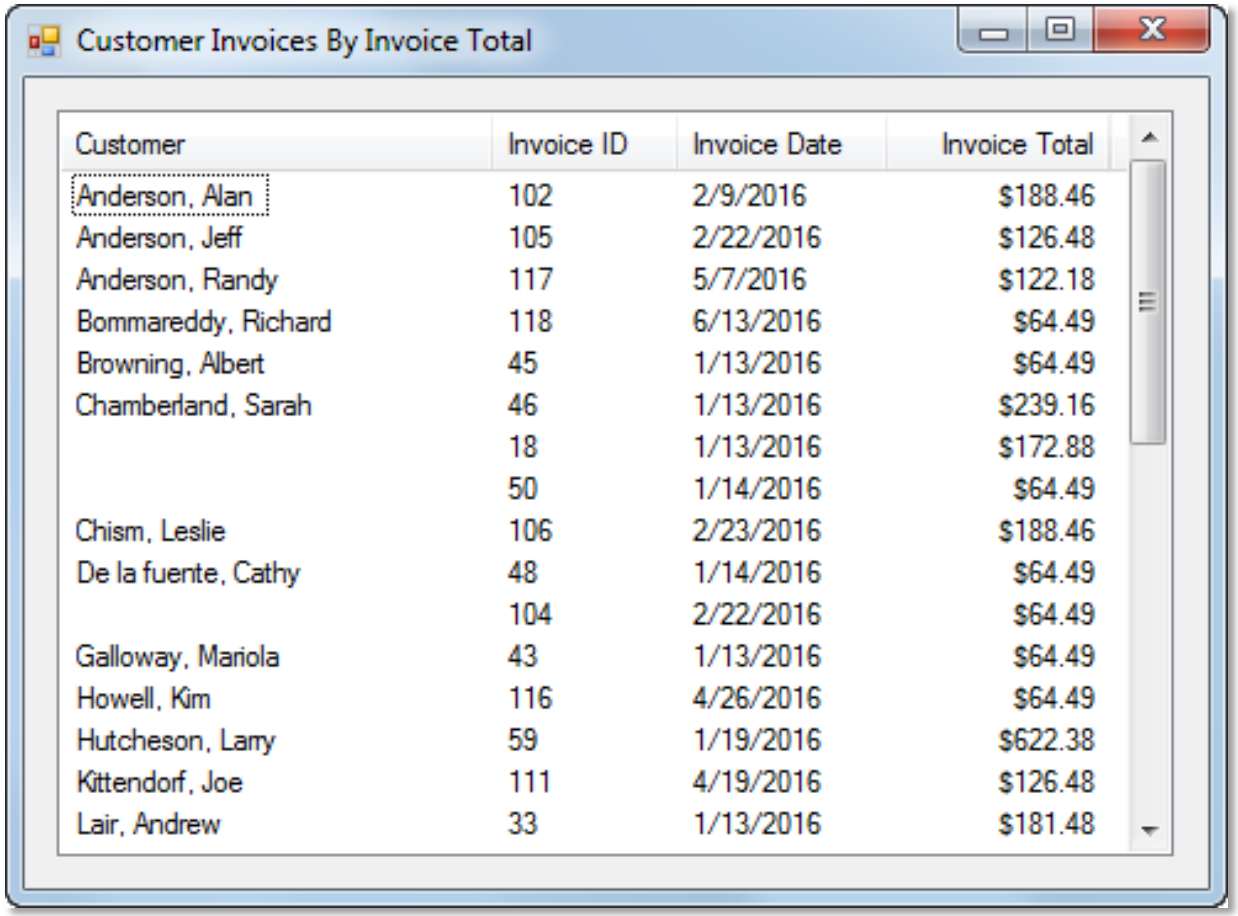
# The resulting dialog box



# A query that uses extension methods and lambda expressions

```
var invoices = invoiceList
               .Where(i => i.InvoiceTotal > 150)
               .OrderBy(i => i.CustomerID)
               .ThenByDescending(i => i.InvoiceTotal)
               .Select(i => new { i.CustomerID,
                                  i.InvoiceTotal });
```

# The Customer Invoice form




| Customer | Invoice ID | Invoice Date | Invoice Total |
|---|---|---|---|
| Anderson, Alan | 102 | 2/9/2016 | $188.46 |
| Anderson, Jeff | 105 | 2/22/2016 | $126.48 |
| Anderson, Randy | 117 | 5/7/2016 | $122.18 |
| Bommareddy, Richard | 118 | 6/13/2016 | $64.49 |
| Browning, Albert | 45 | 1/13/2016 | $64.49 |
| Chamberland, Sarah | 46 | 1/13/2016 | $239.16 |
| | 18 | 1/13/2016 | $172.88 |
| | 50 | 1/14/2016 | $64.49 |
| Chism, Leslie | 106 | 2/23/2016 | $188.46 |
| De la fuente, Cathy | 48 | 1/14/2016 | $64.49 |
| | 104 | 2/22/2016 | $64.49 |
| Galloway, Mariola | 43 | 1/13/2016 | $64.49 |
| Howell, Kim | 116 | 4/26/2016 | $64.49 |
| Hutcheson, Larry | 59 | 1/19/2016 | $622.38 |
| Kittendorf, Joe | 111 | 4/19/2016 | $126.48 |
| Lair, Andrew | 33 | 1/13/2016 | $181.48 |

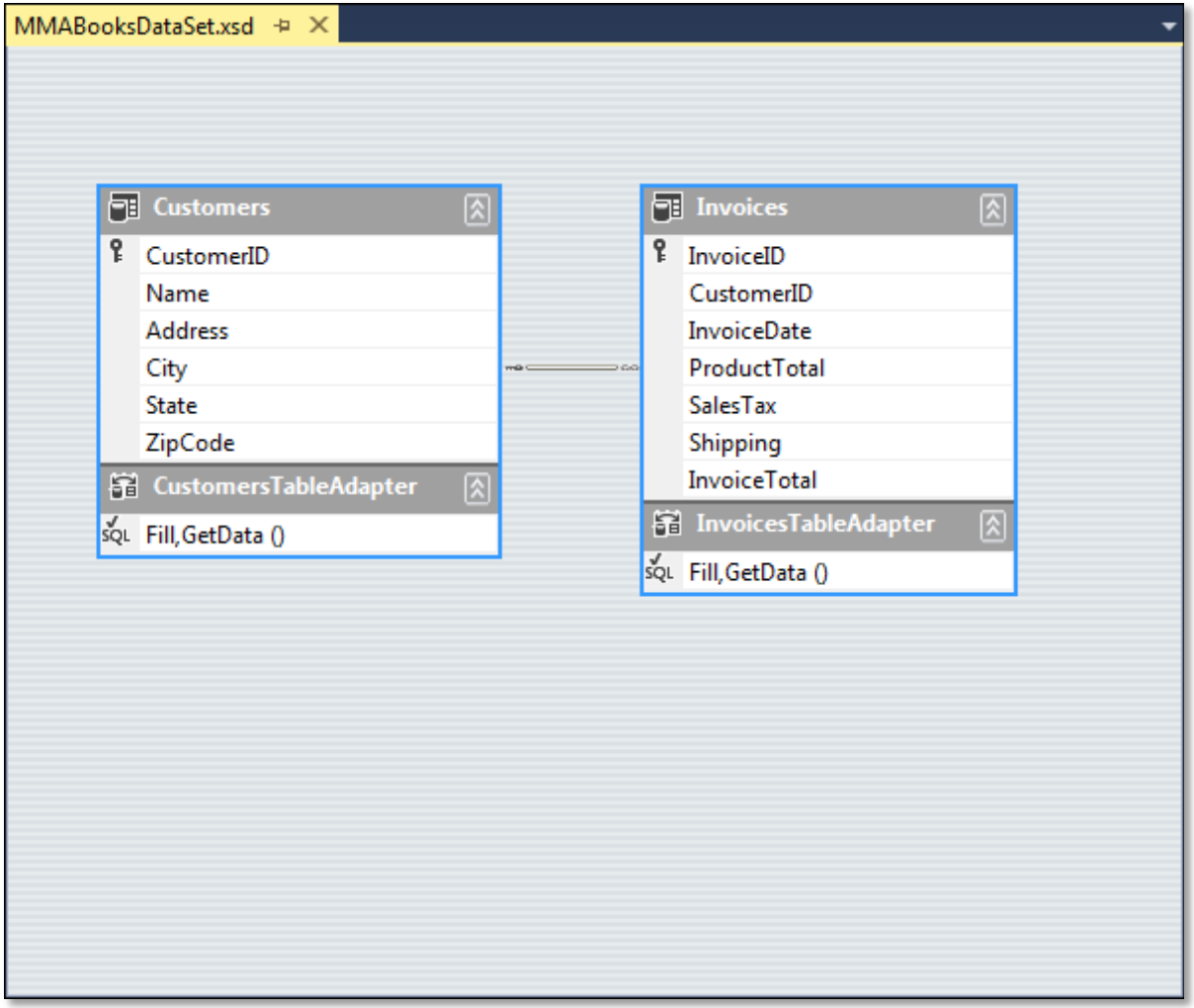
# Customer Invoice form with generic lists

```
private void Form1_Load(object sender, EventArgs e)
{
    List<Customer> customerList =
        CustomerDB.GetCustomers();
    List<Invoice> invoiceList = InvoiceDB.GetInvoices();

    var invoices = from invoice in invoiceList
                   join customer in customerList
                   on invoice.CustomerID
                   equals customer.CustomerID
                   orderby customer.Name,
                           invoice.InvoiceTotal descending
                   select new { customer.Name,
                                invoice.InvoiceID,
                                invoice.InvoiceDate,
                                invoice.InvoiceTotal };

    string customerName = "";
    int i = 0;
```

# Customer Invoice form with generic lists (cont.)

```
    foreach (var invoice in invoices)
    {
        if (invoice.Name != customerName)
        {
            lvInvoices.Items.Add(invoice.Name);
            customerName = invoice.Name;
        }
        else
        {
            lvInvoices.Items.Add("");
        }
        lvInvoices.Items[i].SubItems.Add(
            invoice.InvoiceID.ToString());
        lvInvoices.Items[i].SubItems.Add(
            Convert.ToDateTime(
                invoice.InvoiceDate).ToShortDateString());
        lvInvoices.Items[i].SubItems.Add(
            invoice.InvoiceTotal.ToString("c"));
        i += 1;
    }
}
```

# The dataset schema



# Customer Invoice form with typed dataset

```
MMABooksDataSet mmaBooksDataSet = new MMABooksDataSet();
InvoicesTableAdapter invoicesTableAdapter =
    new InvoicesTableAdapter();
CustomersTableAdapter customersTableAdapter =
    new CustomersTableAdapter();

private void Form1_Load(object sender, EventArgs e)
{
    invoicesTableAdapter.Fill(mmaBooksDataSet.Invoices);
    customersTableAdapter.Fill(mmaBooksDataSet.Customers);

    var invoices = from invoice in mmaBooksDataSet.Invoices
                   join customer in mmaBooksDataSet.Customers
                   on invoice.CustomerID equals customer.CustomerID
                   orderby customer.Name,
                           invoice.InvoiceTotal descending
                   select new { customer.Name,
                                invoice.InvoiceID,
                                invoice.InvoiceDate,
                                invoice.InvoiceTotal };

    string customerName = "";
    int i = 0;
```

# Customer Invoice form with typed dataset (cont.)

```
    foreach (var invoice in invoices)
    {
        if (invoice.Name != customerName)
        {
            lvInvoices.Items.Add(invoice.Name);
            customerName = invoice.Name;
        }
        else
        {
            lvInvoices.Items.Add("");
        }
        lvInvoices.Items[i].SubItems.Add(
            invoice.InvoiceID.ToString());
        lvInvoices.Items[i].SubItems.Add(
            Convert.ToDateTime(
                invoice.InvoiceDate).ToShortDateString());
        lvInvoices.Items[i].SubItems.Add(
            invoice.InvoiceTotal.ToString("c"));
        i += 1;
    }
}
```

Lab: MVC with Entity Framework, doing CRUD

- Create Entity Models
- Use models in the controller classes and views
- Try LINQ
- Try stored procedures to show report

# More optional topics:

- Threading
- Destructor vs Dispose
- Dynamic
- Xamarin

# Final projects plans

- Projects: CRM, Grid game

Will upload basic requirements this weekend.

Discuss more details next week.

Upload codes in the last week.

Present in the last week or/and publish (optionally ).