

How to use the Entity Framework

Objectives

Applied

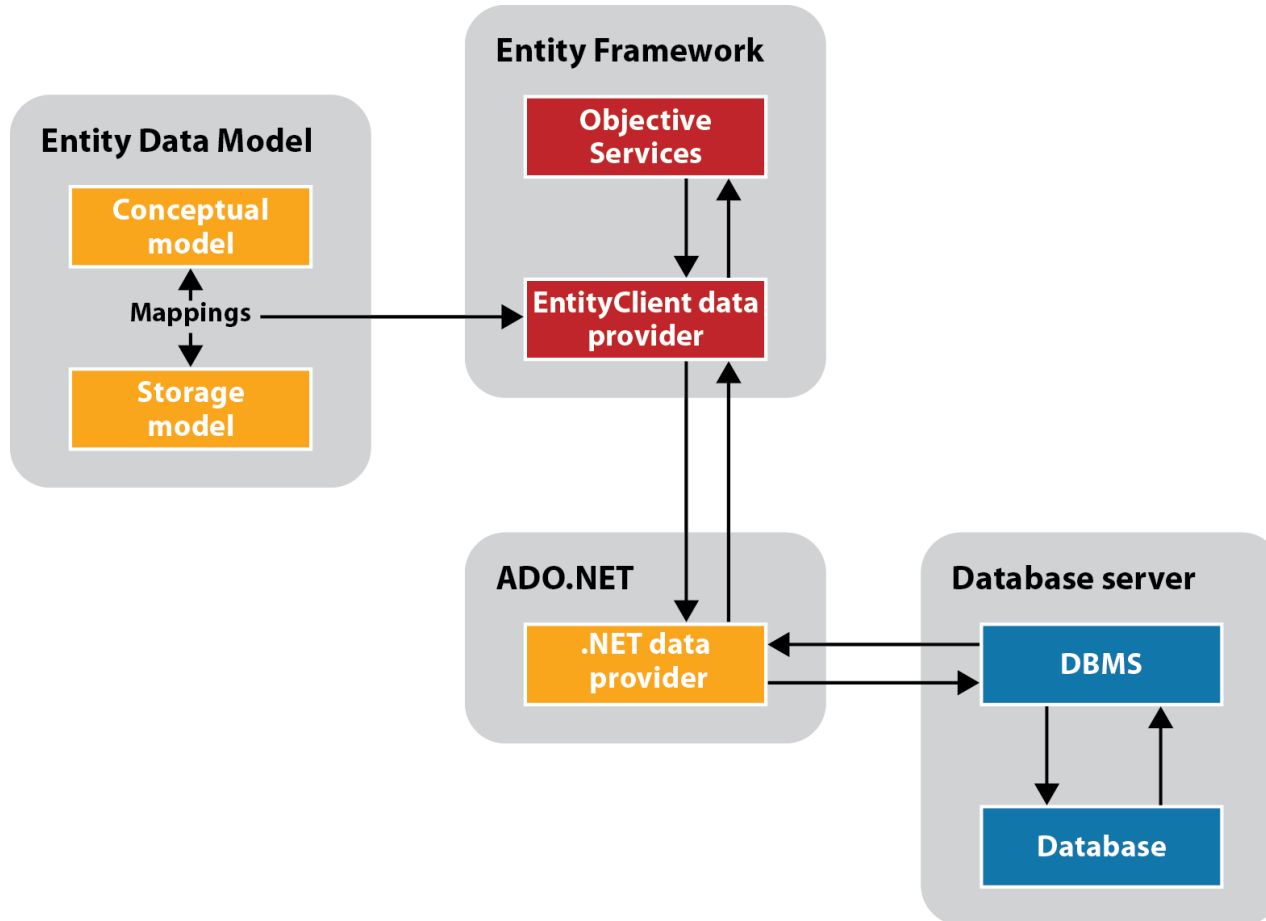
1. Create an Entity Data Model that contains one or more entity classes.
2. Use the Entity Data Model Designer to work with an Entity Data Model.
3. Use LINQ to Entities to retrieve data from one or more database tables.
4. Use the Entity Framework to add, modify, and delete rows in a database table.

Objectives (cont.)

Knowledge

1. In general terms, explain how the Entity Framework works.
2. Describe these terms as they relate to an Entity Data Model: conceptual model, storage model, mappings, object context, entity class, navigation property, and association.
3. Explain how you can provide for concurrency when using the Entity Framework.

How the Entity Framework works



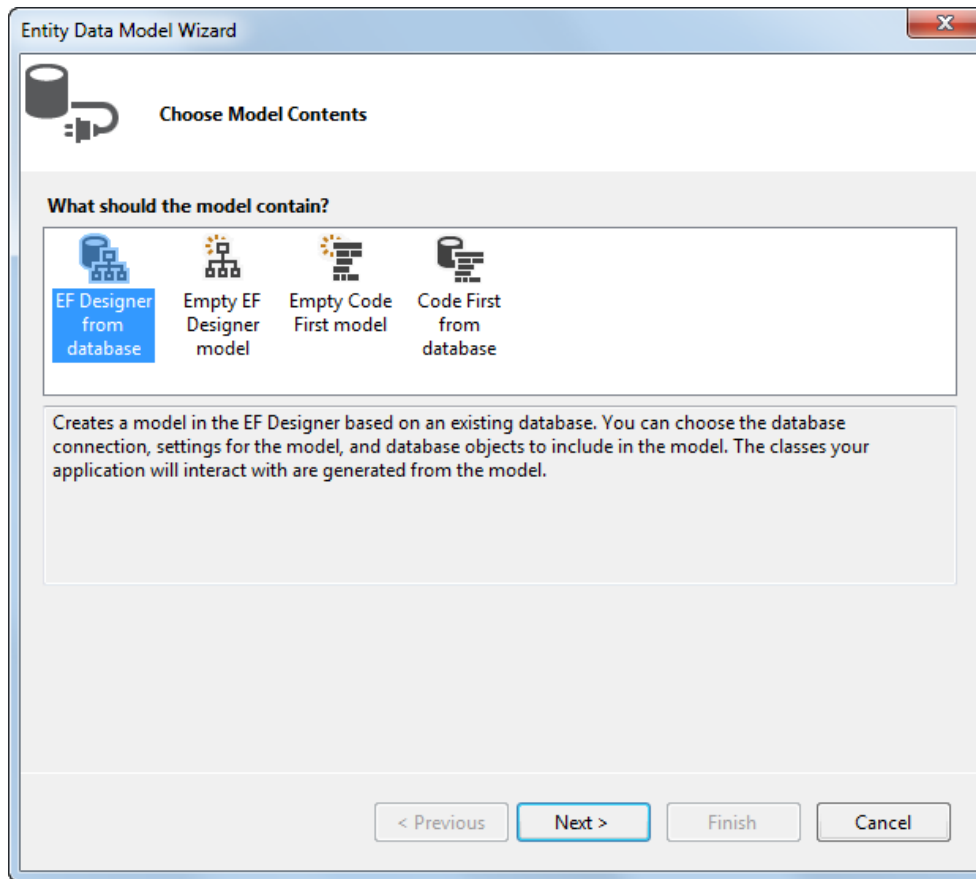
Entity Framework concepts

- 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.

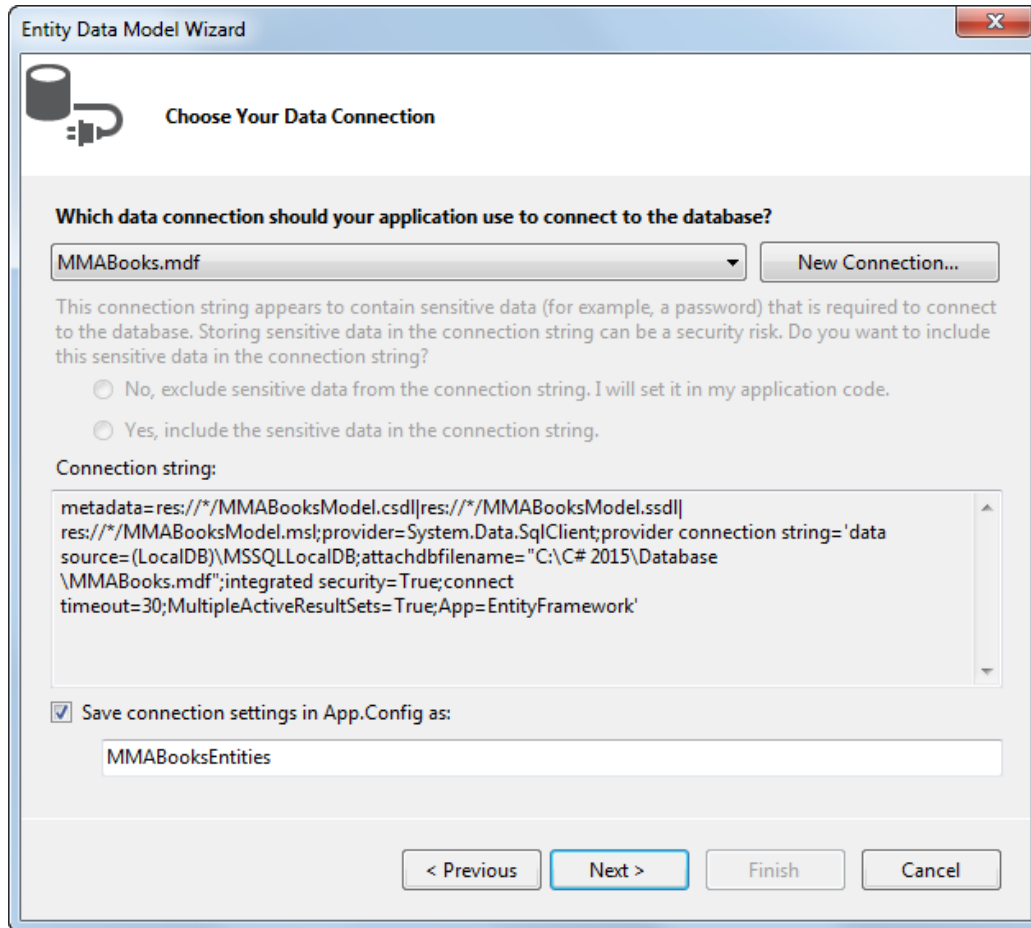
Three ways to query a conceptual model

- LINQ to Entities
- Entity SQL
- Query builder methods

The Entity Data Model Wizard: Step 1




The Entity Data Model Wizard: Step 2



The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Data Connection' step. The window has a title bar with the text 'Entity Data Model Wizard' and a close button. Below the title bar is a header area with a database icon and the text 'Choose Your Data Connection'. The main content area contains a question: 'Which data connection should your application use to connect to the database?'. Below this question is a dropdown menu showing 'MMABooks.mdf' and a 'New Connection...' button. A paragraph of text explains that the connection string might contain sensitive data like a password and asks if the user wants to include it. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' and 'Yes, include the sensitive data in the connection string.'. Below this is a section labeled 'Connection string:' with a text box containing a detailed connection string for a local database. At the bottom, there is a checkbox labeled 'Save connection settings in App.Config as:' which is checked, and a text box containing 'MMABooksEntities'. The bottom of the window has four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'.

Entity Data Model Wizard

 Choose Your Data Connection

Which data connection should your application use to connect to the database?

MMABooks.mdf New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Connection string:

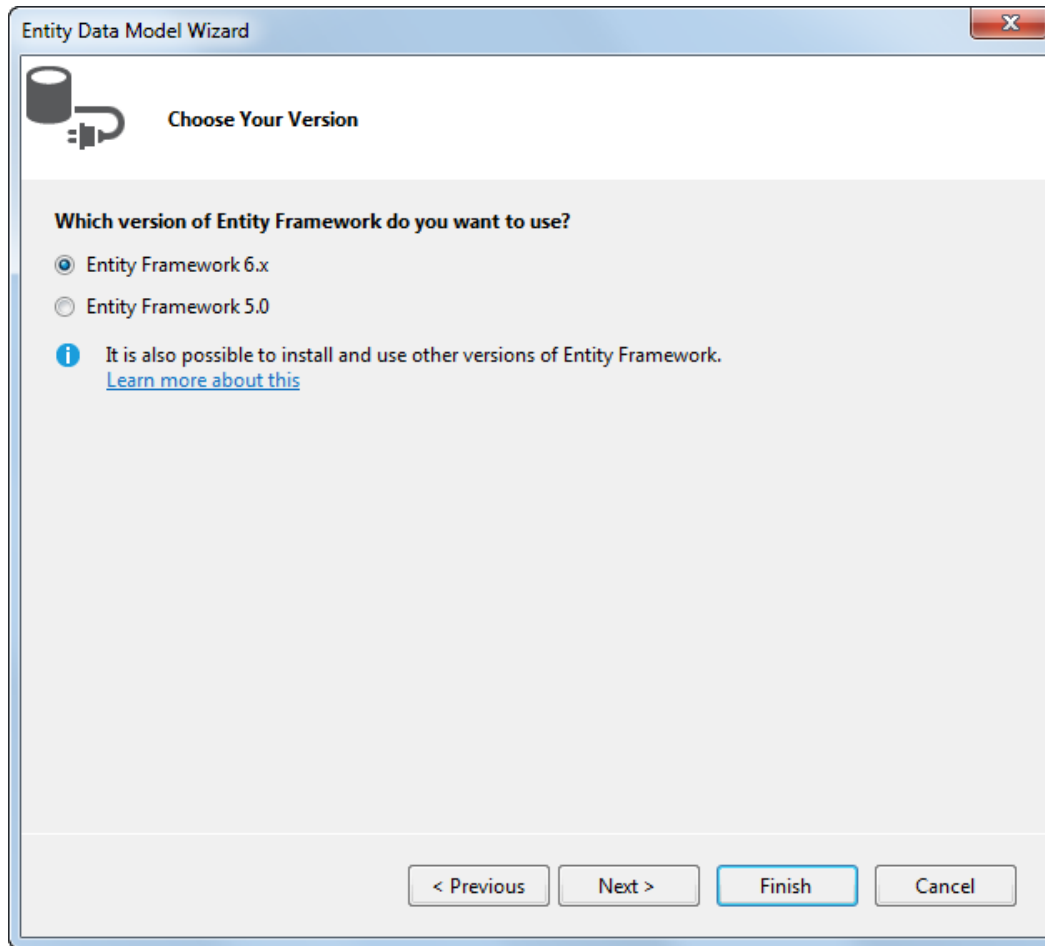
```
metadata=res://*/MMABooksModel.csdl|res://*/MMABooksModel.ssdl|
res://*/MMABooksModel.msl;provider=System.Data.SqlClient;provider connection string='data
source=(LocalDB)\MSSQLLocalDB;attachdbfilename="C:\C# 2015\Database
\MMABooks.mdf";integrated security=True;connect
timeout=30;MultipleActiveResultSets=True;App=EntityFramework'
```

☒ Save connection settings in App.Config as:

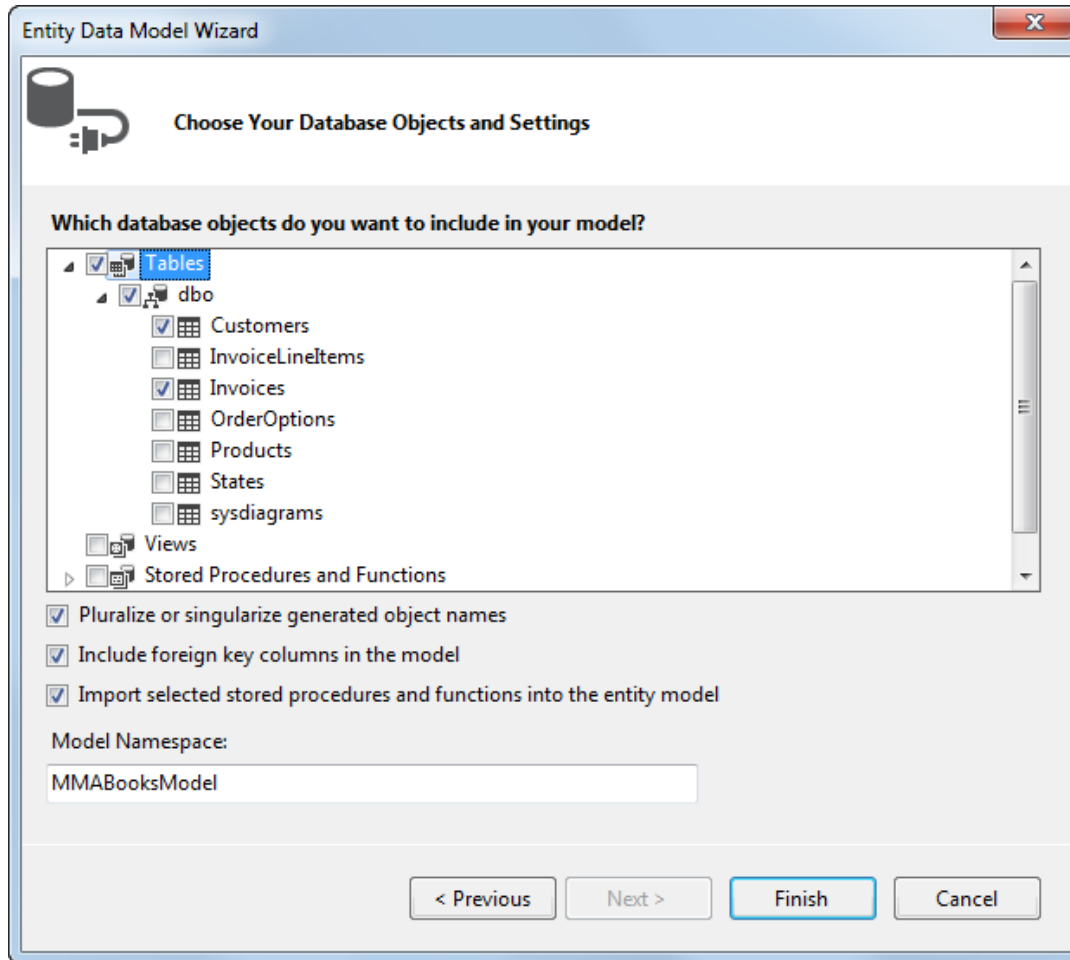
MMABooksEntities

< Previous Next > Finish Cancel

The Entity Data Model Wizard: Step 3



The Entity Data Model Wizard: Step 4



The screenshot shows the 'Entity Data Model Wizard' window at Step 4, titled 'Choose Your Database Objects and Settings'. The window has a standard Windows title bar with a close button. Inside, there's a database icon and the title. The main area asks 'Which database objects do you want to include in your model?'. A tree view shows 'Tables' selected, with a sub-tree for 'dbo' containing 'Customers', 'InvoiceLineItems', 'Invoices', 'OrderOptions', 'Products', 'States', and 'sysdiagrams'. 'Views' and 'Stored Procedures and Functions' are also listed. Below the tree, three checkboxes are checked: 'Pluralize or singularize generated object names', 'Include foreign key columns in the model', and 'Import selected stored procedures and functions into the entity model'. A text box for 'Model Namespace:' contains 'MMABooksModel'. At the bottom are four buttons: '< Previous', 'Next >', 'Finish' (highlighted), and 'Cancel'.

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ Tables
 - ☒ dbo
 - ☒ Customers
 - ☐ InvoiceLineItems
 - ☒ Invoices
 - ☐ OrderOptions
 - ☐ Products
 - ☐ States
 - ☐ sysdiagrams
 - ☐ Views
 - ☐ Stored Procedures and Functions

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

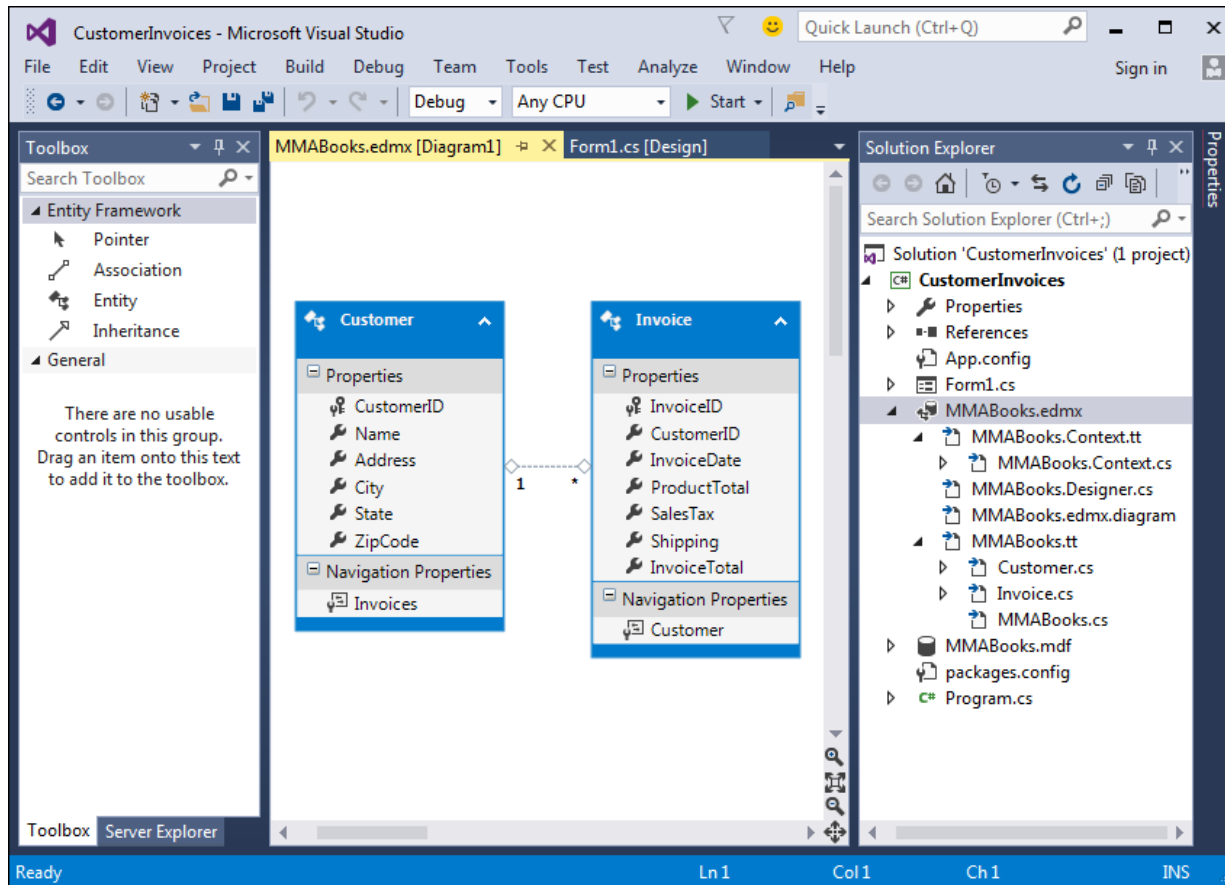
☒ Import selected stored procedures and functions into the entity model

Model Namespace:

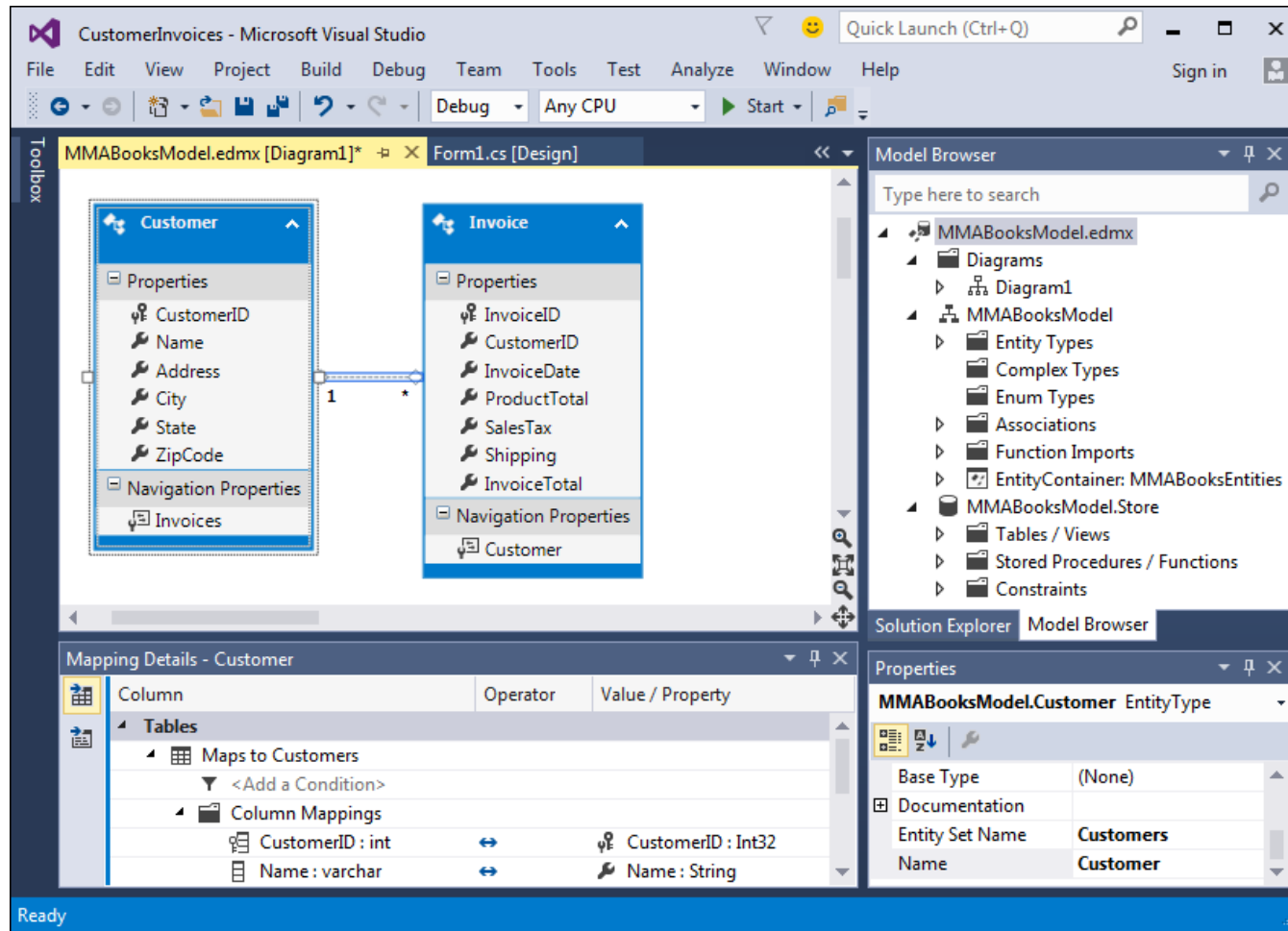
MMABooksModel

< Previous Next > **Finish** Cancel

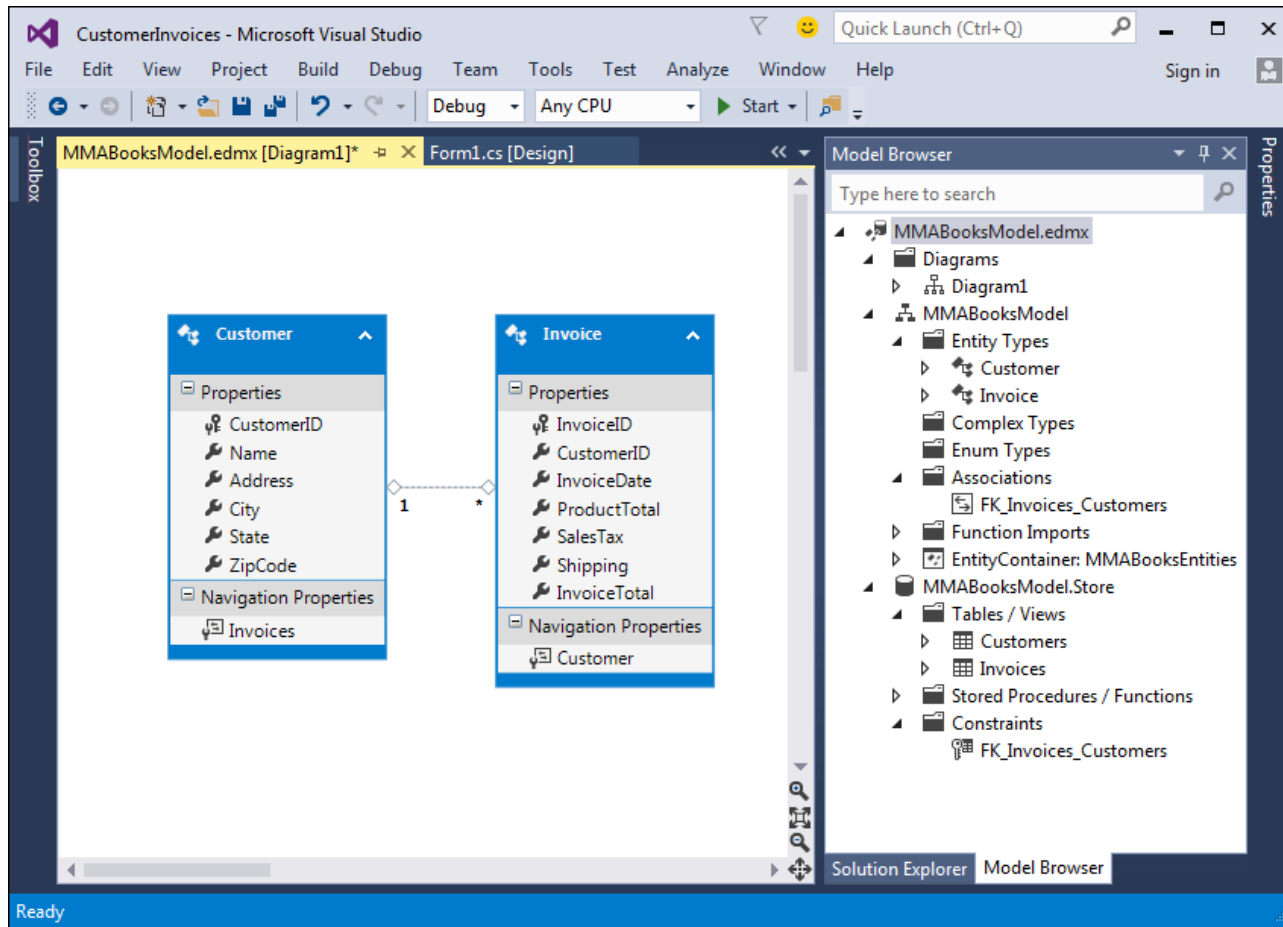
The Entity Data Model in the Entity Data Model Designer



The Entity Data Model Designer



The Model Browser window with some of its nodes expanded



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

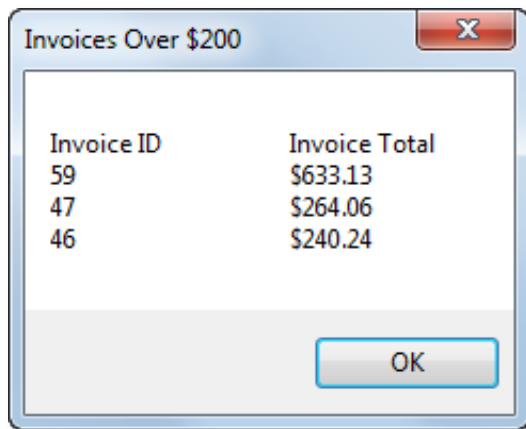
Mapping Details - Invoice			▼	🔍	✕
Column	Operator	Value / Property			
Tables					
Maps to Invoices					
<Add a Condition>					
Column Mappings					
InvoiceID : int	↔	🔑	InvoiceID : Int32		
CustomerID : int	↔	🔑	CustomerID : Int32		
InvoiceDate : datetime	↔	🔑	InvoiceDate : DateTime		
ProductTotal : money	↔	🔑	<Delete>		
SalesTax : money	↔	🔑	CustomerID : Int32		
Shipping : money	↔	🔑	InvoiceDate : DateTime		
InvoiceTotal : money	↔	🔑	InvoiceID : Int32		
<Add a Table or View>		InvoiceTotal : Decimal			
		ProductTotal : Decimal			
		SalesTax : Decimal			
		Shipping : Decimal			

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();
    ...
}
```

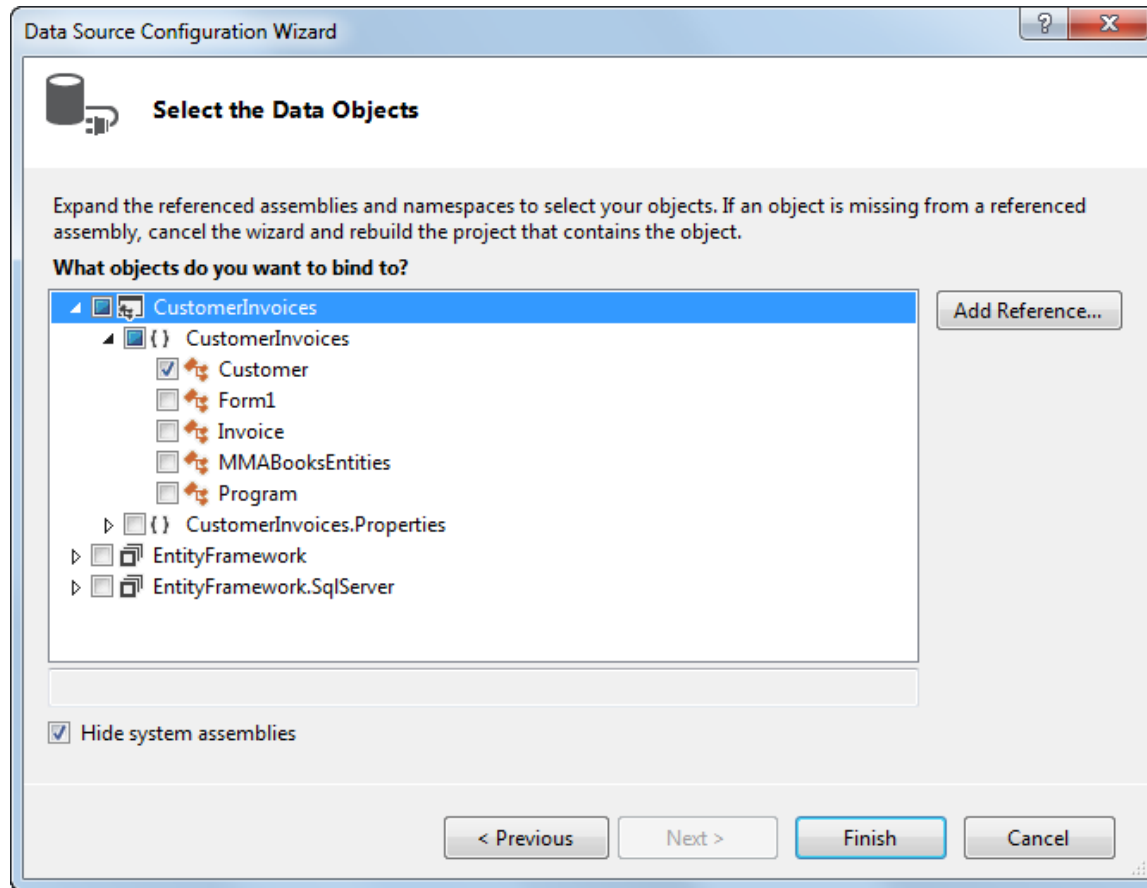
Code in a catch block that uses client wins for concurrency exceptions

```
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";
```

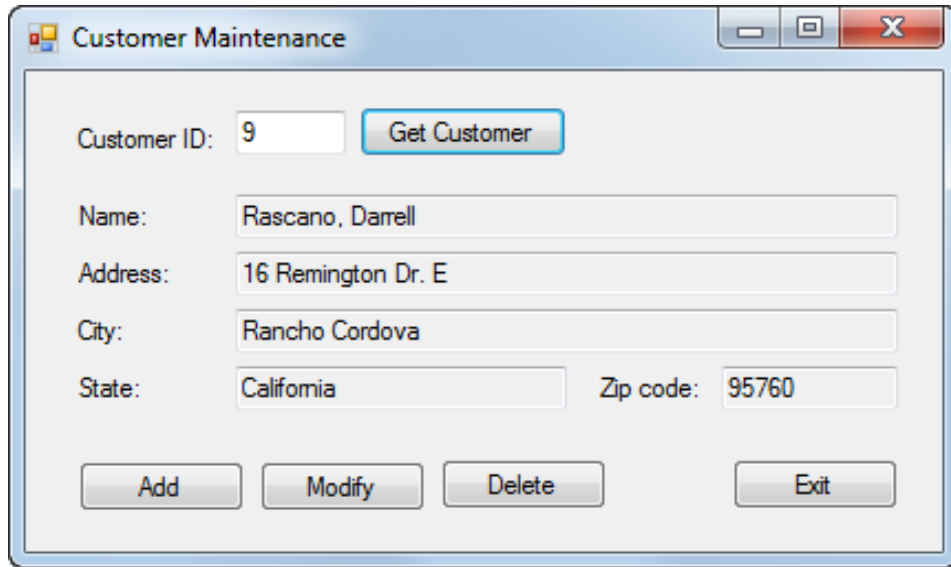
Code that binds a combo box to the results of a query

```
var customers =  
    from customer in mmaBooks.Customers  
    orderby customer.Name  
    select new { customer.CustomerID, customer.Name };  
  
cboCustomers.DataSource = customers.ToList();  
cboCustomers.DisplayMember = "Name";  
cboCustomers.ValueMember = "CustomerID";
```

A statement that binds a combo box using its binding source

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

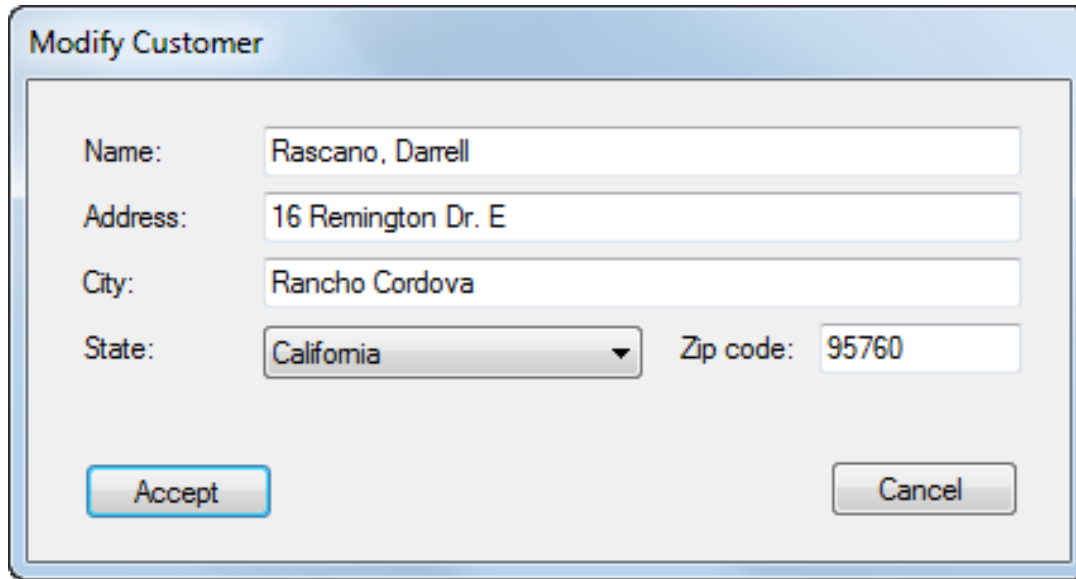
The Customer Maintenance Form



A screenshot of a software window titled "Customer Maintenance". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main area contains several input fields and buttons. At the top, there is a "Customer ID:" label followed by a text box containing the number "9" and a "Get Customer" button. Below this, there are four rows of labels and text boxes: "Name:" with "Rascano, Darrell", "Address:" with "16 Remington Dr. E", "City:" with "Rancho Cordova", and "State:" with "California". To the right of the "State:" field is a "Zip code:" label followed by a text box containing "95760". At the bottom of the window, there are four buttons: "Add", "Modify", "Delete", and "Exit".

Field	Value
Customer ID	9
Name	Rascano, Darrell
Address	16 Remington Dr. E
City	Rancho Cordova
State	California
Zip code	95760

The Add/Modify Customer form

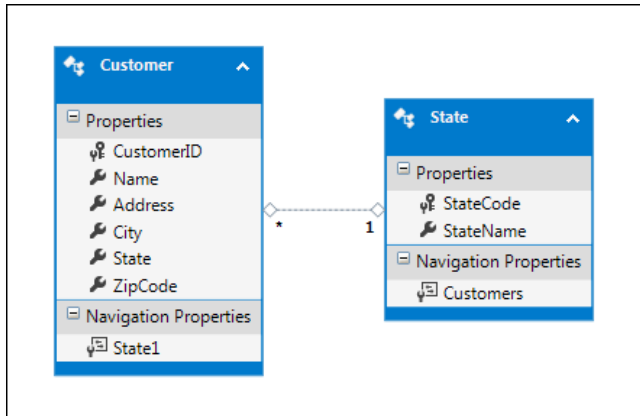


The image shows a software window titled "Modify Customer". It contains a form with the following fields and values:

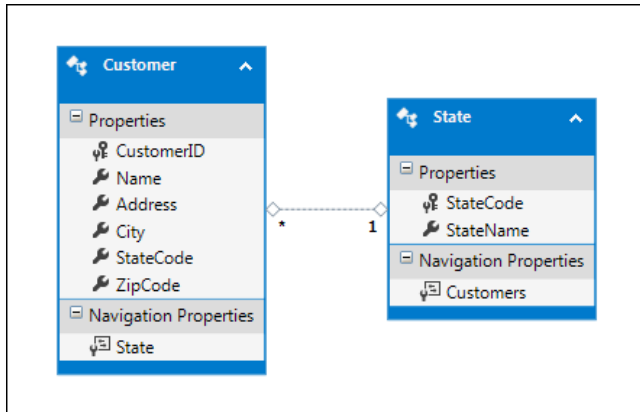
Field	Value
Name:	Rascano, Darrell
Address:	16 Remington Dr. E
City:	Rancho Cordova
State:	California
Zip code:	95760

At the bottom of the form are two buttons: "Accept" and "Cancel".

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.)

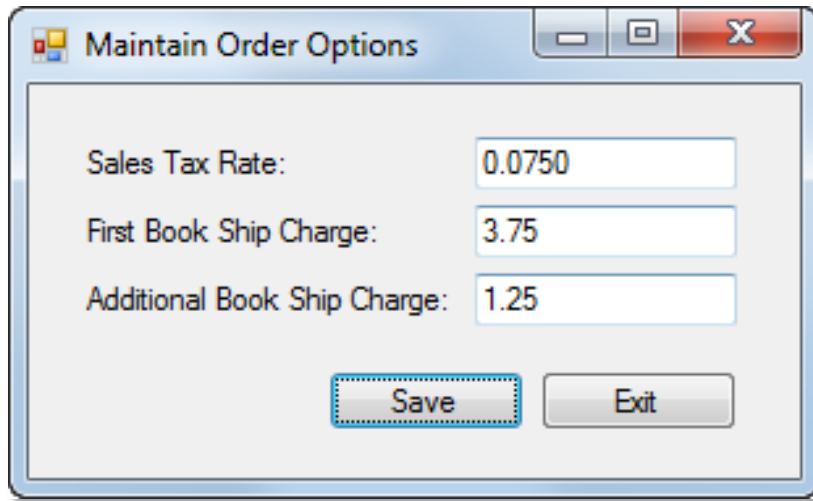
```
        catch (DbUpdateConcurrencyException ex)
        {
            ex.Entries.Single().Reload();
            if (MMABooksEntity.mmaBooks.Entry(customer).State ==
                EntityState.Detached)
            {
                MessageBox.Show("Another user has deleted " +
                    "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);
}

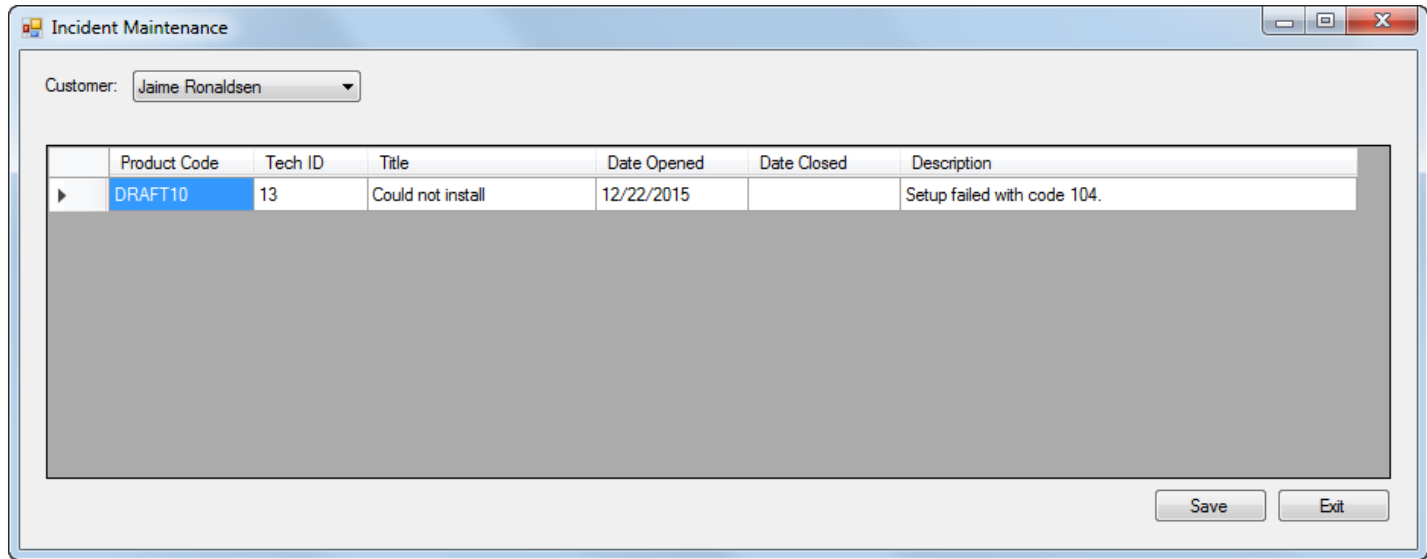
private void PutCustomerData(Customer customer)
{
    customer.Name = txtName.Text;
    customer.Address = txtAddress.Text;
    customer.City = txtCity.Text;
    customer.StateCode = cboStates.SelectedValue.ToString();
    customer.ZipCode = txtZipCode.Text;
}
}
```

Extra 24-1 Use the Entity Framework to create an Order Options Maintenance application



Use the Entity Framework to create an application that lets the user update the data in a table of order options.

Project 5-3 Maintain incidents



The screenshot shows a window titled "Incident Maintenance". At the top, there is a "Customer:" label followed by a dropdown menu showing "Jaime Ronaldsen". Below this is a table with the following columns: Product Code, Tech ID, Title, Date Opened, Date Closed, and Description. The first row of the table is highlighted in blue and contains the following data: Product Code "DRAFT10", Tech ID "13", Title "Could not install", Date Opened "12/22/2015", Date Closed (empty), and Description "Setup failed with code 104." Below the table is a large gray rectangular area. At the bottom right of the window, there are two buttons: "Save" and "Exit".

	Product Code	Tech ID	Title	Date Opened	Date Closed	Description
▶	DRAFT10	13	Could not install	12/22/2015		Setup failed with code 104.

Develop an application that lets the user assign a date closed and change the description for open incidents.