# Создание простого приложения для обработки данных с помощью WPF и Entity Framework 6

**Предупреждение**

В этом пошаговом руководстве показано, как создать базовое приложение "формы по данным" в Visual Studio. Приложение использует SQL Server LocalDB, базу данных Northwind, Entity Framework 6 (не Entity Framework Core) и Windows Presentation Foundation для платформа .NET Framework (не .NET Core или .NET 5 или более поздней версии). Здесь показано, как выполнять базовую привязку данных с помощью представления основных сведений, а также настраиваемый навигатор привязок с кнопками **"Переместить далее**", **"Переместить назад**", **"Перейти к началу**", " **Переместить в конец**", " **Обновить** " и **"Удалить**".

В этой статье основное внимание уделяется использованию средств обработки данных в Visual Studio и не пытается подробно объяснить базовые технологии. Предполагается, что у вас есть базовое знакомство с XAML, Entity Framework и SQL. В этом примере также не демонстрируется архитектура Model-View-ViewModel (MVVM), которая является стандартной для приложений WPF. Однако этот код можно скопировать в собственное приложение MVVM с небольшим количеством изменений.

Окончательный код для этого руководства можно найти в GitHub на сайте VS Tutorial Samples — EF6.

## Установка и подключение к Northwind

В этом примере используется SQL Server Express LocalDB и пример базы данных Northwind. Если поставщик данных ADO.NET для этого продукта поддерживает Entity Framework, он также должен работать с другими продуктами базы данных SQL.

1. Если у вас нет SQL Server Express LocalDB, установите его с помощью **Visual Studio Installer**. В **Visual Studio Installer** можно установить SQL Server Express LocalDB как часть рабочей нагрузки **хранилища данных и обработки** или как отдельный компонент.
2. Установите образец базы данных Northwind, выполнив следующие действия.
   1. В Visual Studio откройте окно **SQL Server обозреватель объектов**. (**SQL Server обозреватель объектов** устанавливается как часть рабочей нагрузки **хранилища данных и обработки** в **Visual Studio Installer**.) Разверните узел **SQL Server**. Щелкните правой кнопкой мыши экземпляр LocalDB и выберите **"Создать запрос**".

Откроется окно редактора запросов.

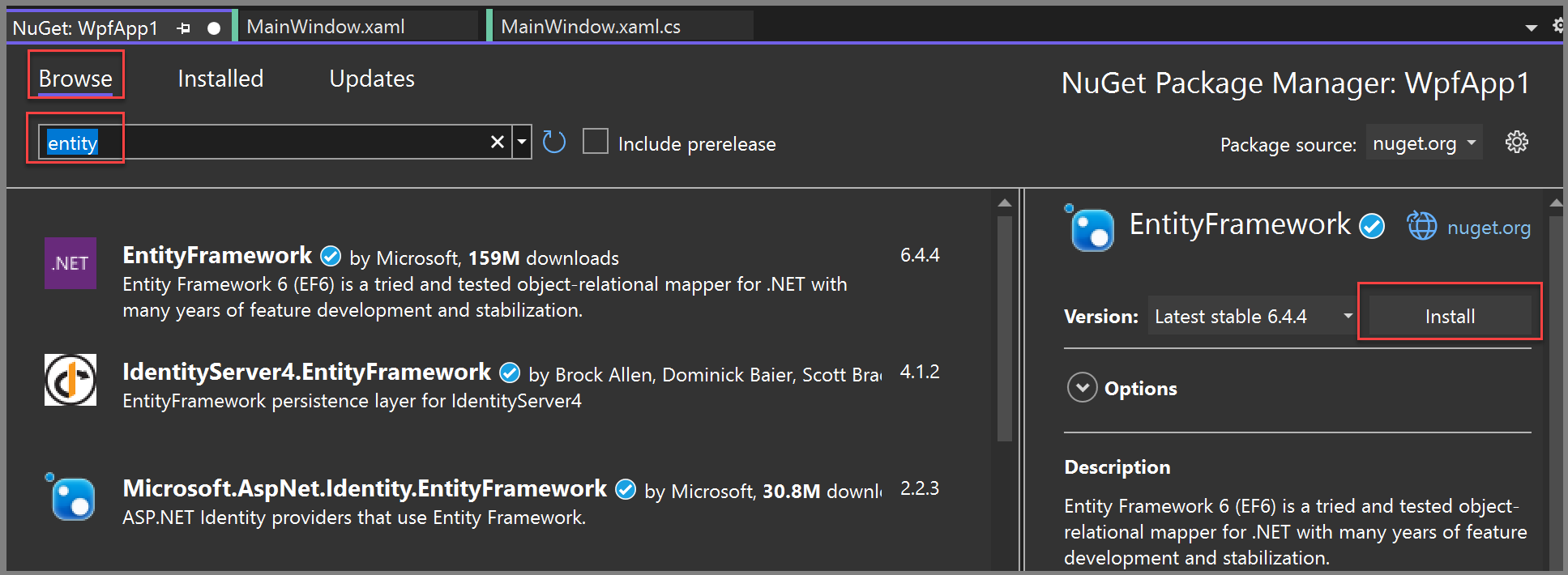
* 1. Скопируйте скрипт Transact-SQL Northwind в буфер обмена. Этот скрипт T-SQL создает базу данных Northwind с нуля и заполняет ее данными.
  2. Вставьте скрипт T-SQL в редактор запросов и нажмите кнопку **"Выполнить** ".

Через некоторое время запрос завершится и база данных Northwind будет создана.

1. Добавьте новые подключения для Northwind.

## Настройка проекта

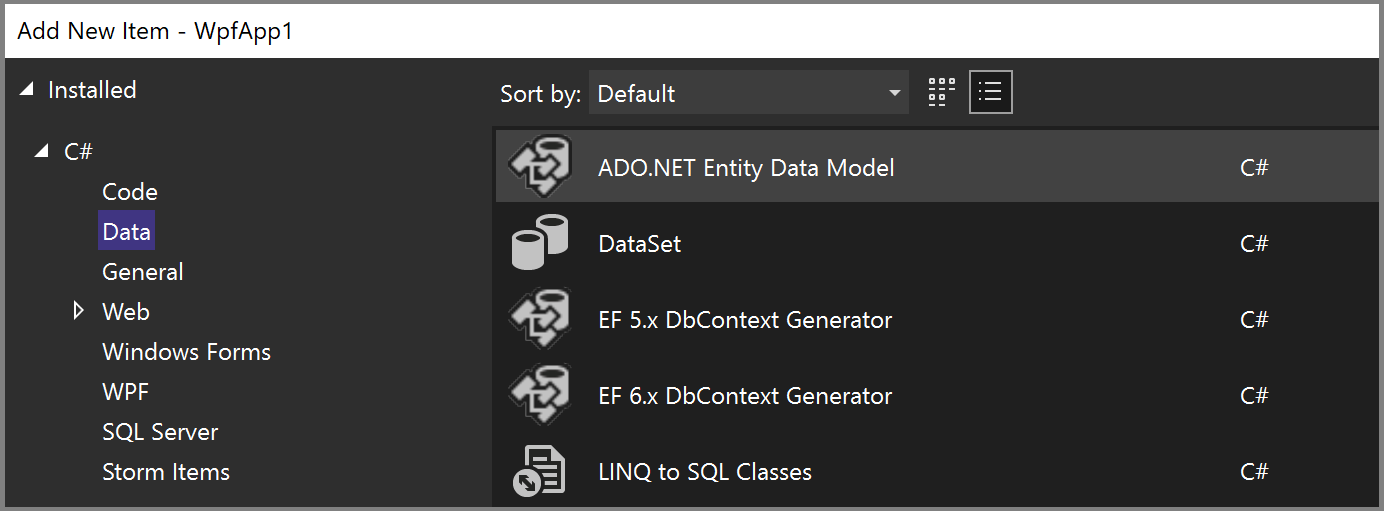
1. В Visual Studio создайте проект **приложения WPF на C# (платформа .NET Framework).**
2. Добавьте пакет NuGet для Entity Framework 6. В **Обозреватель решений** выберите узел проекта. В главном меню выберите "**Управление пакетами NuGet" проекта**>.
3. В **диспетчере пакетов NuGet** щелкните ссылку **"Обзор** ". Entity Framework, вероятно, является верхним пакетом в списке. Нажмите кнопку **"Установить** " в правой области и следуйте инструкциям. Окно вывода сообщает о завершении установки.



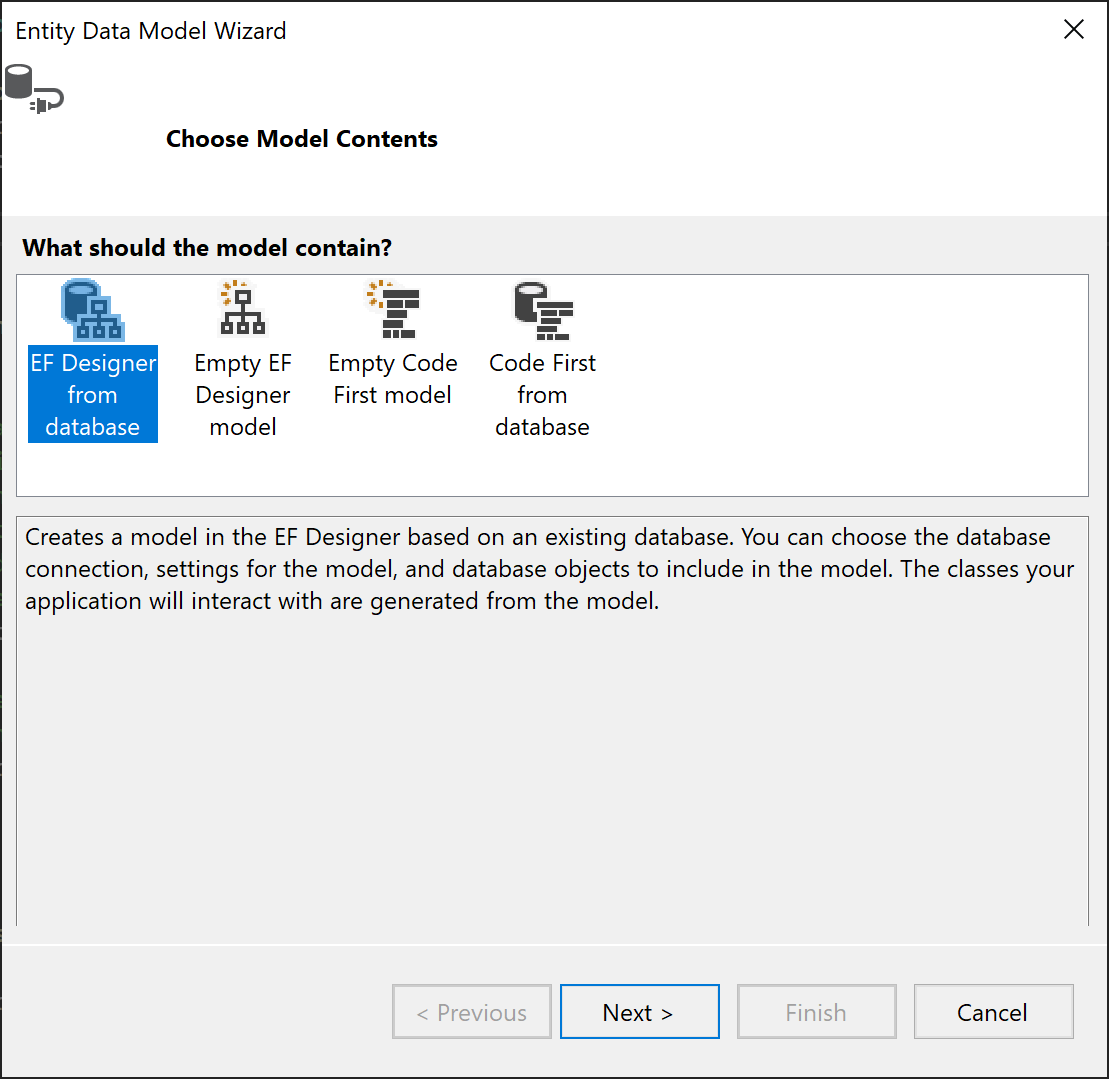
1. Теперь вы можете использовать Visual Studio для создания модели на основе базы данных Northwind.

## Создание модели

1. Щелкните правой кнопкой мыши узел проекта в **обозревателе решений** и последовательно выберите **Добавить**>**Новый элемент**. В левой области в узле C# выберите **"Данные** " и в средней области выберите **ADO.NET Entity Data Model**.



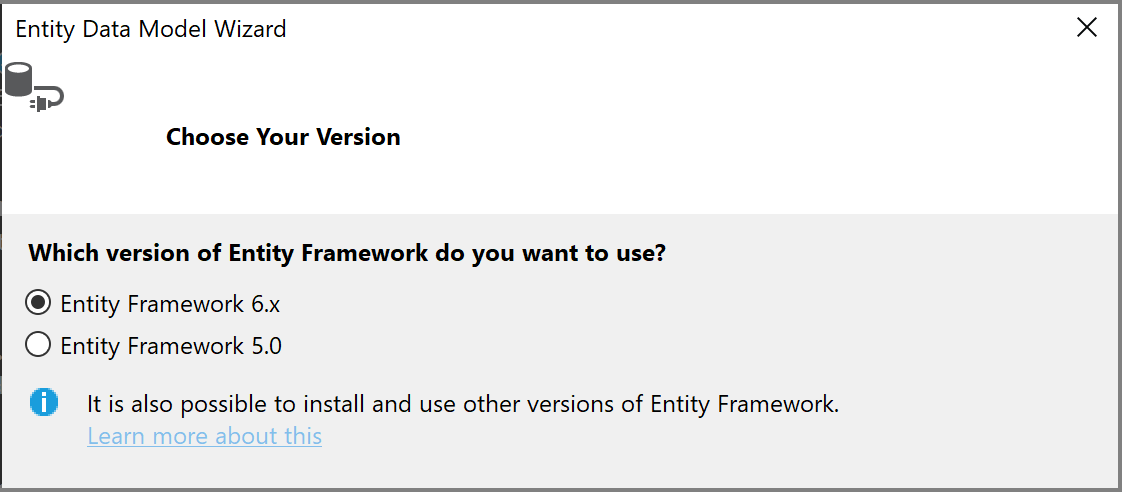
1. Вызовите модель Northwind\_model и нажмите кнопку **"Добавить**". **Откроется мастер модели данных сущностей**. Выберите **конструктор EF из базы данных** и нажмите кнопку **"Далее**".



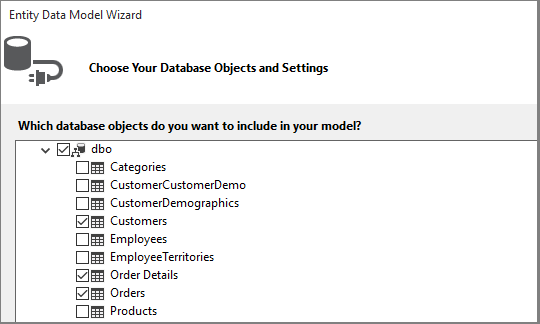
1. На следующем экране выберите подключение LocalDB Northwind (например, (localdb)\MSSQLLocalDB), укажите базу данных Northwind и нажмите кнопку **"Далее**".

Если подключение не отображается, нажмите кнопку **"Создать подключение**", а затем в диалоговом окне **"Выбор источника данных**" выберите **"Microsoft SQL Server**", нажмите кнопку **"Продолжить**" и в диалоговом окне **"Свойства подключения**", введите (localdb)\MSSQLLocalDB и в разделе "**Выбор или ввод имени базы данных**", нажмите кнопку **"Northwind**", а затем нажмите кнопку **"ОК**".

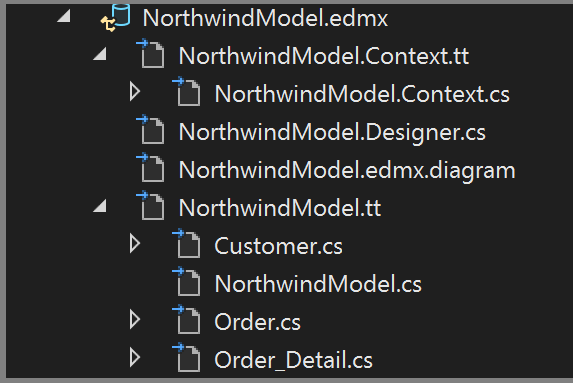
1. При появлении запроса выберите используемую версию Entity Framework.



1. На следующей странице мастера выберите таблицы, хранимые процедуры и другие объекты базы данных для включения в модель Entity Framework. Разверните узел dbo в представлении дерева и выберите **"Клиенты**", " **Заказы**" и **"Сведения о заказе**". Оставьте флажки по умолчанию и нажмите кнопку **"Готово**".



1. Мастер создает классы C#, представляющие модель Entity Framework. Классы являются обычными старыми классами C#, и они являются тем, что мы привязали к пользовательскому интерфейсу WPF. EDMX-файл описывает связи и другие метаданные, которые связывают классы с объектами в базе данных. TT-файлы — это шаблоны T4, которые создают код, который работает с моделью и сохраняет изменения в базе данных. Все эти файлы можно просмотреть в **Обозреватель решений** в узле Northwind\_model:



Область конструктора для EDMX-файла позволяет изменять некоторые свойства и связи в модели. Мы не будем использовать конструктор в этом пошаговом руководстве.

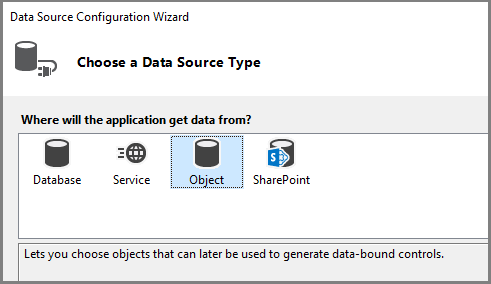
1. TT-файлы являются общими целями и необходимо настроить один из них для работы с привязкой данных WPF, для чего требуется ObservableCollections. В **Обозреватель решений** разверните узел Northwind\_model, пока не найдете Northwind\_model.tt. (Убедитесь, что вы не находитесь в CONTEXT.TT-файле , который находится непосредственно под EDMX-файлом .)
   * Замените два вхождения ICollection на ObservableCollection<T>.
   * Замените первое вхождение HashSet<T>ObservableCollection<T> вокруг строки 51. Не заменяйте второе вхождение HashSet.
   * Замените единственное вхождение System.Collections.Generic (вокруг строки 431) System.Collections.ObjectModelна .
2. Нажмите **клавишу F5** или **CTRL**+**F5** , чтобы создать и запустить проект. При первом запуске приложения классы модели видны мастеру источников данных.

Теперь вы готовы подключить эту модель к странице XAML, чтобы можно было просматривать, перемещаться и изменять данные.

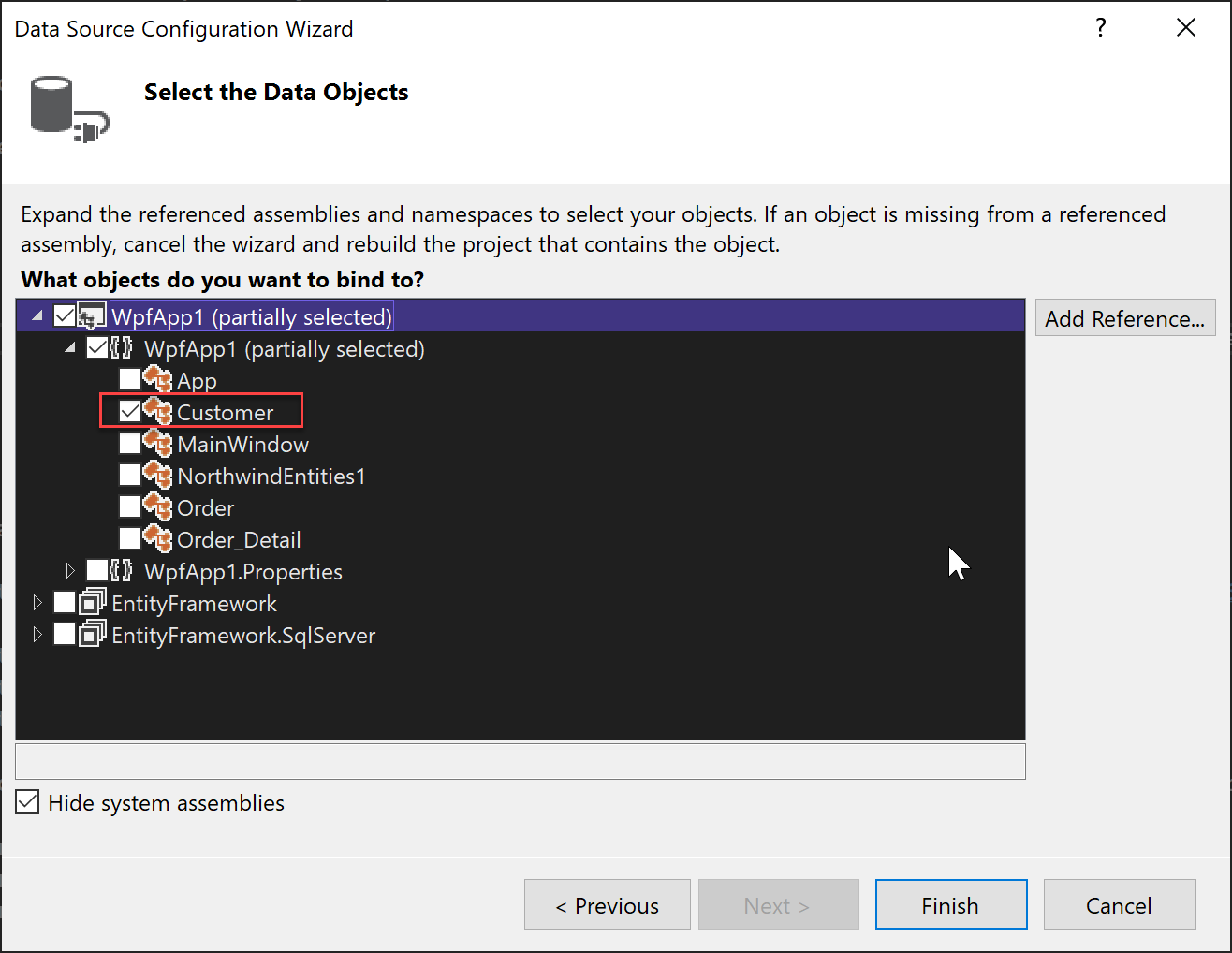
## Привязка модели к странице XAML

Можно написать собственный код привязки данных, но гораздо проще позволить Visual Studio сделать это самостоятельно.

1. В главном меню выберите "Добавить**новый источник данных"**>, чтобы открыть **мастер настройки источника данных**. Выберите **объект** , так как вы привязываются к классам модели, а не к базе данных:



1. Разверните узел проекта и выберите **"Клиент**". (Источники для заказов автоматически создаются из свойства навигации Orders в Customer.)



1. Нажмите кнопку **Готово**.
2. Перейдите к MainWindow.xaml в представлении кода. Мы сохраняем простой КОД XAML для целей этого примера. Измените заголовок MainWindow на что-то более описательное и увеличьте его высоту и ширину до 600 x 800 на данный момент. Вы всегда можете изменить его позже. Теперь добавьте эти три определения строк в основную сетку, одну строку для кнопок навигации, одну для сведений о клиенте и одну для сетки, которая показывает свои заказы:

XAML

<Grid.RowDefinitions>

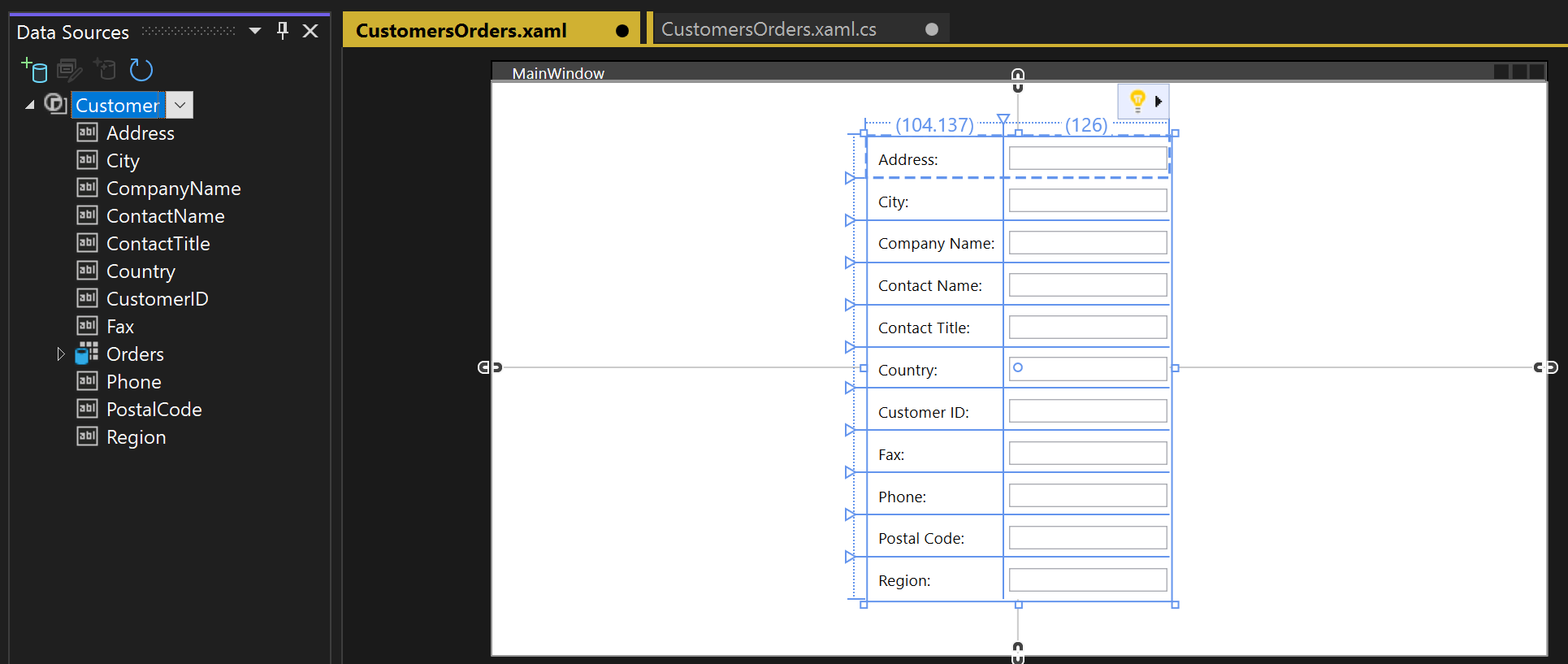
<RowDefinition Height="auto"/>

<RowDefinition Height="auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

1. Теперь откройте файл MainWindow.xaml , чтобы просмотреть его в конструкторе. Это приводит к тому, что окно **"Источники данных** " отображается как параметр в поле окна Visual Studio рядом с **панелью элементов**. Щелкните вкладку, чтобы открыть окно, или нажмите **клавиши SHIFT ALT**++**D** или выберите **"Просмотреть**> другие**источники данныхWindows**>". Мы будем отображать каждое свойство в классе Customers в отдельном текстовом поле. Сначала щелкните стрелку в поле со списком **"Клиенты"** и выберите **"Сведения**". Затем перетащите узел в середину области конструктора, чтобы конструктор знал, что он будет отображаться в средней строке. Если она неуместна, вы можете указать строку вручную в XAML (Grid.Row="1"). По умолчанию элементы управления помещаются по вертикали в элемент сетки, но на этом этапе их можно расположить в форме. Например, можно поместить текстовое поле **"Имя"** сверху над адресом. Пример приложения для этой статьи переупорядочение полей и их переупорядочение на два столбца.

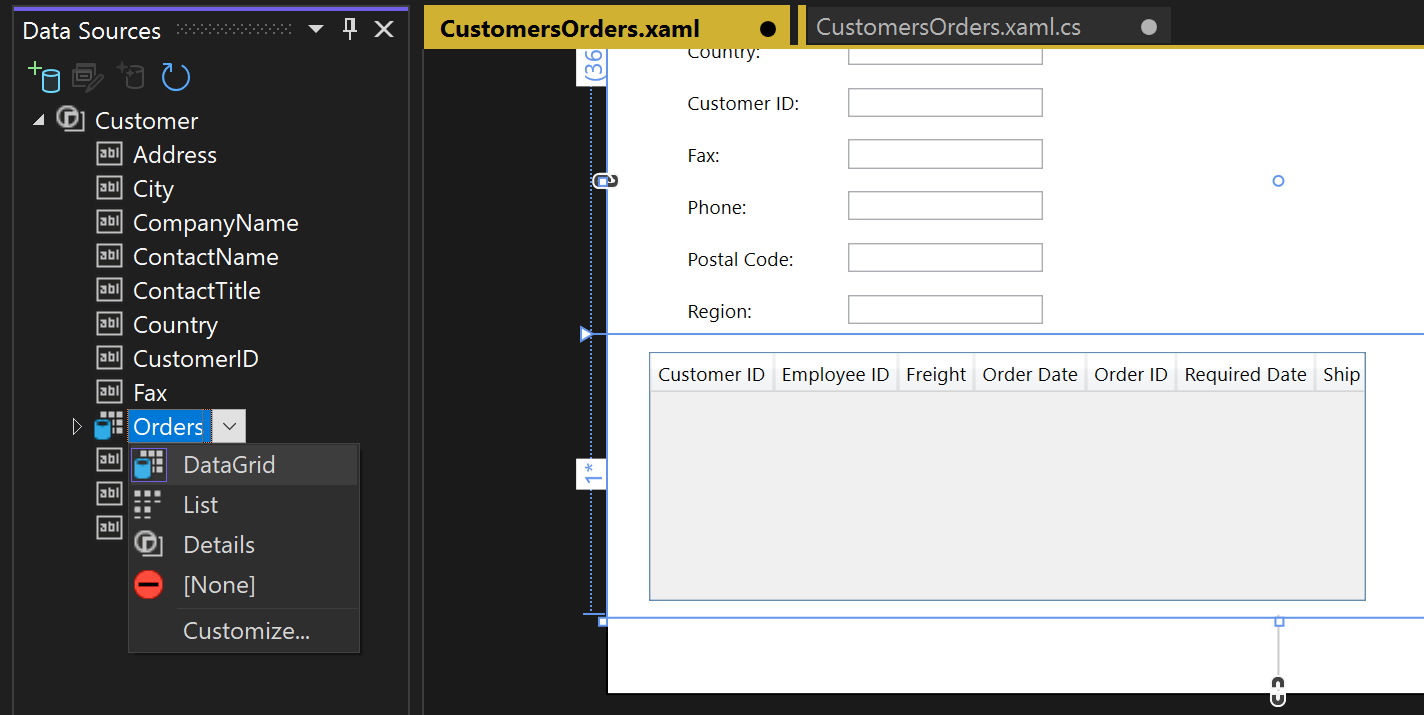


В представлении XAML теперь можно увидеть новый Grid элемент в строке 1 (средней строке) родительской сетки. Родительская сетка имеет DataContext атрибут, ссылающийся на CollectionViewSource элемент, добавленный в Windows.Resources элемент. Учитывая, что контекст данных, когда первое текстовое поле привязывается к **Address**, это имя сопоставляется со Address свойством в текущем Customer объекте в объекте CollectionViewSource.

XAML

<Grid DataContext="{StaticResource customerViewSource}">

1. Когда клиент отображается в верхней половине окна, вы хотите увидеть свои заказы в нижней половине. Вы отображаете заказы в одном элементе управления представления сетки. Чтобы привязка данных master-detail работала должным образом, важно привязать свойство Orders в классе Customers, а не к отдельному узлу Orders. Перетащите свойство Orders класса Customers в нижнюю половину формы, чтобы конструктор положил его в строку 2:



1. Visual Studio создал весь код привязки, который соединяет элементы управления пользовательского интерфейса с событиями в модели. Все, что нужно сделать, чтобы просмотреть некоторые данные, — написать код для заполнения модели. Сначала перейдите к MainWindow.xaml.cs и добавьте элемент данных в класс MainWindow для контекста данных. Этот объект, созданный для вас, действует как элемент управления, который отслеживает изменения и события в модели. Вы также добавите элементы данных CollectionViewSource для клиентов и заказов, а также связанную логику инициализации конструктора в существующий конструктор MainWindow(). Верхняя часть класса должна выглядеть следующим образом:

C#

public partial class MainWindow : Window

{

NorthwindEntities context = new NorthwindEntities();

CollectionViewSource custViewSource;

CollectionViewSource ordViewSource;

public MainWindow()

{

InitializeComponent();

custViewSource = ((CollectionViewSource)(FindResource("customerViewSource")));

ordViewSource = ((CollectionViewSource)(FindResource("customerOrdersViewSource")));

DataContext = this;

}

Если он еще не существует, добавьте директиву using для System.Data.Entity, чтобы перенести Load метод расширения в область действия:

C#

using System.Data.Entity;

Теперь прокрутите страницу вниз и найдите Window\_Loaded обработчик событий. Обратите внимание, что Visual Studio добавил объект CollectionViewSource. Это представляет объект NorthwindEntities, выбранный при создании модели. Вы добавили это уже, поэтому вам не нужно его здесь. Давайте заменим код Window\_Loaded таким образом, чтобы метод выглядел следующим образом:

C#

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

// Load is an extension method on IQueryable,

// defined in the System.Data.Entity namespace.

// This method enumerates the results of the query,

// similar to ToList but without creating a list.

// When used with Linq to Entities, this method

// creates entity objects and adds them to the context.

context.Customers.Load();

// After the data is loaded, call the DbSet<T>.Local property

// to use the DbSet<T> as a binding source.

custViewSource.Source = context.Customers.Local;

}

1. Нажмите клавишу **F5**. Вы увидите сведения о первом клиенте, который был получен в CollectionViewSource. Кроме того, их заказы должны отображаться в сетке данных. Форматирование не очень велико, поэтому давайте исправим это. Вы также можете создать способ просмотра других записей и выполнения базовых операций CRUD.

## Настройка макета страницы и добавление сетки для новых клиентов и заказов

Расположение по умолчанию, созданное Visual Studio, не идеально подходит для вашего приложения, поэтому мы предоставим окончательный код XAML для копирования в код. Вам также нужны некоторые формы (которые фактически являются сетками), чтобы позволить пользователю добавить нового клиента или заказ. Чтобы добавить нового клиента и заказ, вам потребуется отдельный набор текстовых полей, которые не привязаны к CollectionViewSourceданным. Вы будете управлять сеткой, которую пользователь видит в любой момент времени, задав свойство Visible в методах обработчика. Наконец, вы добавите кнопку "Удалить" в каждую строку в сетке "Заказы", чтобы пользователь мог удалить отдельный заказ.

Сначала добавьте эти стили в Windows.Resources элемент MainWindow.xaml:

XAML

<Style x:Key="Label" TargetType="{x:Type Label}" BasedOn="{x:Null}">

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="Margin" Value="3"/>

<Setter Property="Height" Value="23"/>

</Style>

<Style x:Key="CustTextBox" TargetType="{x:Type TextBox}" BasedOn="{x:Null}">

<Setter Property="HorizontalAlignment" Value="Right"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="Margin" Value="3"/>

<Setter Property="Height" Value="26"/>

<Setter Property="Width" Value="120"/>

</Style>

Затем замените всю внешнюю сетку этой разметкой:

XAMLКопировать

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="auto"/>

<RowDefinition Height="auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid x:Name="existingCustomerGrid" Grid.Row="1" HorizontalAlignment="Left" Margin="5" Visibility="Visible" VerticalAlignment="Top" Background="AntiqueWhite" DataContext="{StaticResource customerViewSource}">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" MinWidth="233"/>

<ColumnDefinition Width="Auto" MinWidth="397"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Content="Customer ID:" Grid.Row="0" Style="{StaticResource Label}"/>

<TextBox x:Name="customerIDTextBox" Grid.Row="0" Style="{StaticResource CustTextBox}"

Text="{Binding CustomerID, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Company Name:" Grid.Row="1" Style="{StaticResource Label}"/>

<TextBox x:Name="companyNameTextBox" Grid.Row="1" Style="{StaticResource CustTextBox}"

Text="{Binding CompanyName, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Contact Name:" Grid.Row="2" Style="{StaticResource Label}"/>

<TextBox x:Name="contactNameTextBox" Grid.Row="2" Style="{StaticResource CustTextBox}"

Text="{Binding ContactName, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Contact title:" Grid.Row="3" Style="{StaticResource Label}"/>

<TextBox x:Name="contactTitleTextBox" Grid.Row="3" Style="{StaticResource CustTextBox}"

Text="{Binding ContactTitle, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Address:" Grid.Row="4" Style="{StaticResource Label}"/>

<TextBox x:Name="addressTextBox" Grid.Row="4" Style="{StaticResource CustTextBox}"

Text="{Binding Address, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="City:" Grid.Column="1" Grid.Row="0" Style="{StaticResource Label}"/>

<TextBox x:Name="cityTextBox" Grid.Column="1" Grid.Row="0" Style="{StaticResource CustTextBox}"

Text="{Binding City, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Country:" Grid.Column="1" Grid.Row="1" Style="{StaticResource Label}"/>

<TextBox x:Name="countryTextBox" Grid.Column="1" Grid.Row="1" Style="{StaticResource CustTextBox}"

Text="{Binding Country, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Fax:" Grid.Column="1" Grid.Row="2" Style="{StaticResource Label}"/>

<TextBox x:Name="faxTextBox" Grid.Column="1" Grid.Row="2" Style="{StaticResource CustTextBox}"

Text="{Binding Fax, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Phone:" Grid.Column="1" Grid.Row="3" Style="{StaticResource Label}"/>

<TextBox x:Name="phoneTextBox" Grid.Column="1" Grid.Row="3" Style="{StaticResource CustTextBox}"

Text="{Binding Phone, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Postal Code:" Grid.Column="1" Grid.Row="4" VerticalAlignment="Center" Style="{StaticResource Label}"/>

<TextBox x:Name="postalCodeTextBox" Grid.Column="1" Grid.Row="4" Style="{StaticResource CustTextBox}"

Text="{Binding PostalCode, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Region:" Grid.Column="1" Grid.Row="5" Style="{StaticResource Label}"/>

<TextBox x:Name="regionTextBox" Grid.Column="1" Grid.Row="5" Style="{StaticResource CustTextBox}"

Text="{Binding Region, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

</Grid>

<Grid x:Name="newCustomerGrid" Grid.Row="1" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="5" DataContext="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type Window}}, Path=newCustomer, UpdateSourceTrigger=Explicit}" Visibility="Collapsed" Background="CornflowerBlue">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" MinWidth="233"/>

<ColumnDefinition Width="Auto" MinWidth="397"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Content="Customer ID:" Grid.Row="0" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_customerIDTextBox" Grid.Row="0" Style="{StaticResource CustTextBox}"

Text="{Binding CustomerID, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Company Name:" Grid.Row="1" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_companyNameTextBox" Grid.Row="1" Style="{StaticResource CustTextBox}"

Text="{Binding CompanyName, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true }"/>

<Label Content="Contact Name:" Grid.Row="2" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_contactNameTextBox" Grid.Row="2" Style="{StaticResource CustTextBox}"

Text="{Binding ContactName, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Contact title:" Grid.Row="3" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_contactTitleTextBox" Grid.Row="3" Style="{StaticResource CustTextBox}"

Text="{Binding ContactTitle, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Address:" Grid.Row="4" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_addressTextBox" Grid.Row="4" Style="{StaticResource CustTextBox}"

Text="{Binding Address, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="City:" Grid.Column="1" Grid.Row="0" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_cityTextBox" Grid.Column="1" Grid.Row="0" Style="{StaticResource CustTextBox}"

Text="{Binding City, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Country:" Grid.Column="1" Grid.Row="1" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_countryTextBox" Grid.Column="1" Grid.Row="1" Style="{StaticResource CustTextBox}"

Text="{Binding Country, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Fax:" Grid.Column="1" Grid.Row="2" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_faxTextBox" Grid.Column="1" Grid.Row="2" Style="{StaticResource CustTextBox}"

Text="{Binding Fax, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Phone:" Grid.Column="1" Grid.Row="3" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_phoneTextBox" Grid.Column="1" Grid.Row="3" Style="{StaticResource CustTextBox}"

Text="{Binding Phone, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Postal Code:" Grid.Column="1" Grid.Row="4" VerticalAlignment="Center" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_postalCodeTextBox" Grid.Column="1" Grid.Row="4" Style="{StaticResource CustTextBox}"

Text="{Binding PostalCode, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Region:" Grid.Column="1" Grid.Row="5" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_regionTextBox" Grid.Column="1" Grid.Row="5" Style="{StaticResource CustTextBox}"

Text="{Binding Region, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

</Grid>

<Grid x:Name="newOrderGrid" Grid.Row="1" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="5" DataContext="{Binding Path=newOrder, Mode=TwoWay}" Visibility="Collapsed" Background="LightGreen">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" MinWidth="233"/>

<ColumnDefinition Width="Auto" MinWidth="397"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Content="New Order Form" FontWeight="Bold"/>

<Label Content="Employee ID:" Grid.Row="1" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_employeeIDTextBox" Grid.Row="1" Style="{StaticResource CustTextBox}"

Text="{Binding EmployeeID, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Order Date:" Grid.Row="2" Style="{StaticResource Label}"/>

<DatePicker x:Name="add\_orderDatePicker" Grid.Row="2" HorizontalAlignment="Right" Width="120"

SelectedDate="{Binding OrderDate, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>

<Label Content="Required Date:" Grid.Row="3" Style="{StaticResource Label}"/>

<DatePicker x:Name="add\_requiredDatePicker" Grid.Row="3" HorizontalAlignment="Right" Width="120"

SelectedDate="{Binding RequiredDate, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>

<Label Content="Shipped Date:" Grid.Row="4" Style="{StaticResource Label}"/>

<DatePicker x:Name="add\_shippedDatePicker" Grid.Row="4" HorizontalAlignment="Right" Width="120"

SelectedDate="{Binding ShippedDate, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>

<Label Content="Ship Via:" Grid.Row="5" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_ShipViaTextBox" Grid.Row="5" Style="{StaticResource CustTextBox}"

Text="{Binding ShipVia, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

<Label Content="Freight" Grid.Row="6" Style="{StaticResource Label}"/>

<TextBox x:Name="add\_freightTextBox" Grid.Row="6" Style="{StaticResource CustTextBox}"

Text="{Binding Freight, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}"/>

</Grid>

<DataGrid x:Name="ordersDataGrid" SelectionUnit="Cell" SelectionMode="Single" AutoGenerateColumns="False" CanUserAddRows="false" IsEnabled="True" EnableRowVirtualization="True" Width="auto" ItemsSource="{Binding Source={StaticResource customerOrdersViewSource}}" Margin="10,10,10,10" Grid.Row="2" RowDetailsVisibilityMode="VisibleWhenSelected">

<DataGrid.Columns>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<Button Content="Delete" Command="{StaticResource DeleteOrderCommand}" CommandParameter="{Binding}"/>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

<DataGridTextColumn x:Name="customerIDColumn" Binding="{Binding CustomerID}" Header="Customer ID" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="employeeIDColumn" Binding="{Binding EmployeeID}" Header="Employee ID" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="freightColumn" Binding="{Binding Freight}" Header="Freight" Width="SizeToHeader"/>

<DataGridTemplateColumn x:Name="orderDateColumn" Header="Order Date" Width="SizeToHeader">

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<DatePicker SelectedDate="{Binding OrderDate, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

<DataGridTextColumn x:Name="orderIDColumn" Binding="{Binding OrderID}" Header="Order ID" Width="SizeToHeader"/>

<DataGridTemplateColumn x:Name="requiredDateColumn" Header="Required Date" Width="SizeToHeader">

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<DatePicker SelectedDate="{Binding RequiredDate, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

<DataGridTextColumn x:Name="shipAddressColumn" Binding="{Binding ShipAddress}" Header="Ship Address" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="shipCityColumn" Binding="{Binding ShipCity}" Header="Ship City" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="shipCountryColumn" Binding="{Binding ShipCountry}" Header="Ship Country" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="shipNameColumn" Binding="{Binding ShipName}" Header="Ship Name" Width="SizeToHeader"/>

<DataGridTemplateColumn x:Name="shippedDateColumn" Header="Shipped Date" Width="SizeToHeader">

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<DatePicker SelectedDate="{Binding ShippedDate, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true, UpdateSourceTrigger=PropertyChanged}"/>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

<DataGridTextColumn x:Name="shipPostalCodeColumn" Binding="{Binding ShipPostalCode}" Header="Ship Postal Code" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="shipRegionColumn" Binding="{Binding ShipRegion}" Header="Ship Region" Width="SizeToHeader"/>

<DataGridTextColumn x:Name="shipViaColumn" Binding="{Binding ShipVia}" Header="Ship Via" Width="SizeToHeader"/>

</DataGrid.Columns>

</DataGrid>

</Grid>

## Добавление кнопок для навигации, добавления, обновления и удаления

В Windows Forms приложениях вы получаете объект BindingNavigator с кнопками для навигации по строкам в базе данных и выполнению базовых операций CRUD. WPF не предоставляет BindingNavigator, но его достаточно просто создать. Это можно сделать с помощью кнопок внутри горизонтального StackPanel и связать кнопки с командами, привязанными к методам в коде программной части.

Логика команд имеет четыре части: (1) команды, (2) привязки, (3) кнопки и (4) обработчики команд в коде программной части.

### Добавление команд, привязок и кнопок в XAML

1. Сначала добавьте команды в файл MainWindow.xaml внутри Windows.Resources элемента:

XAML

<RoutedUICommand x:Key="FirstCommand" Text="First"/>

<RoutedUICommand x:Key="LastCommand" Text="Last"/>

<RoutedUICommand x:Key="NextCommand" Text="Next"/>

<RoutedUICommand x:Key="PreviousCommand" Text="Previous"/>

<RoutedUICommand x:Key="DeleteCustomerCommand" Text="Delete Customer"/>

<RoutedUICommand x:Key="DeleteOrderCommand" Text="Delete Order"/>

<RoutedUICommand x:Key="UpdateCommand" Text="Update"/>

<RoutedUICommand x:Key="AddCommand" Text="Add"/>

<RoutedUICommand x:Key="CancelCommand" Text="Cancel"/>

1. CommandBinding сопоставляет RoutedUICommand событие с методом в коде программной части. Добавьте этот CommandBindings элемент после закрывающего тега Windows.Resources :

XAML

<Window.CommandBindings>

<CommandBinding Command="{StaticResource FirstCommand}" Executed="FirstCommandHandler"/>

<CommandBinding Command="{StaticResource LastCommand}" Executed="LastCommandHandler"/>

<CommandBinding Command="{StaticResource NextCommand}" Executed="NextCommandHandler"/>

<CommandBinding Command="{StaticResource PreviousCommand}" Executed="PreviousCommandHandler"/>

<CommandBinding Command="{StaticResource DeleteCustomerCommand}" Executed="DeleteCustomerCommandHandler"/>

<CommandBinding Command="{StaticResource DeleteOrderCommand}" Executed="DeleteOrderCommandHandler"/>

<CommandBinding Command="{StaticResource UpdateCommand}" Executed="UpdateCommandHandler"/>

<CommandBinding Command="{StaticResource AddCommand}" Executed="AddCommandHandler"/>

<CommandBinding Command="{StaticResource CancelCommand}" Executed="CancelCommandHandler"/>

</Window.CommandBindings>

1. Теперь добавьте кнопки навигации StackPanel , добавления, удаления и обновления. Сначала добавьте этот стиль в:Windows.Resources

XAML

<Style x:Key="NavButton" TargetType="{x:Type Button}" BasedOn="{x:Null}">

<Setter Property="FontSize" Value="24"/>

<Setter Property="FontFamily" Value="Segoe UI Symbol"/>

<Setter Property="Margin" Value="2,2,2,0"/>

<Setter Property="Width" Value="40"/>

<Setter Property="Height" Value="auto"/>

</Style>

Во-вторых, вставьте этот код сразу после RowDefinitions внешнего Grid элемента в начало страницы XAML:

XAML

<StackPanel Orientation="Horizontal" Margin="2,2,2,0" Height="36" VerticalAlignment="Top" Background="Gainsboro" DataContext="{StaticResource customerViewSource}" d:LayoutOverrides="LeftMargin, RightMargin, TopMargin, BottomMargin">

<Button Name="btnFirst" Content="|◄" Command="{StaticResource FirstCommand}" Style="{StaticResource NavButton}"/>

<Button Name="btnPrev" Content="◄" Command="{StaticResource PreviousCommand}" Style="{StaticResource NavButton}"/>

<Button Name="btnNext" Content="►" Command="{StaticResource NextCommand}" Style="{StaticResource NavButton}"/>

<Button Name="btnLast" Content="►|" Command="{StaticResource LastCommand}" Style="{StaticResource NavButton}"/>

<Button Name="btnDelete" Content="Delete Customer" Command="{StaticResource DeleteCustomerCommand}" FontSize="11" Width="120" Style="{StaticResource NavButton}"/>

<Button Name="btnAdd" Content="New Customer" Command="{StaticResource AddCommand}" FontSize="11" Width="80" Style="{StaticResource NavButton}"/>

<Button Content="New Order" Name="btnNewOrder" FontSize="11" Width="80" Style="{StaticResource NavButton}" Click="NewOrder\_click"/>

<Button Name="btnUpdate" Content="Commit" Command="{StaticResource UpdateCommand}" FontSize="11" Width="80" Style="{StaticResource NavButton}"/>

<Button Content="Cancel" Name="btnCancel" Command="{StaticResource CancelCommand}" FontSize="11" Width="80" Style="{StaticResource NavButton}"/>

</StackPanel>

### Добавление обработчиков команд в класс MainWindow

Код программной части является минимальным, за исключением методов добавления и удаления. Навигация выполняется путем вызова методов в свойстве View объекта CollectionViewSource. В этом DeleteOrderCommandHandler разделе показано, как выполнить каскадное удаление в порядке. Сначала необходимо удалить Order\_Details, связанные с ним. Он UpdateCommandHandler добавляет нового клиента или заказ в коллекцию или просто обновляет существующий клиент или заказ с изменениями, сделанными пользователем в текстовых полях.

Добавьте эти методы обработчика в класс MainWindow.xaml.cs. Если в таблице CollectionViewSource для customers есть другое имя, необходимо изменить имя в каждом из этих методов:

C#

private void LastCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

custViewSource.View.MoveCurrentToLast();

}

private void PreviousCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

custViewSource.View.MoveCurrentToPrevious();

}

private void NextCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

custViewSource.View.MoveCurrentToNext();

}

private void FirstCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

custViewSource.View.MoveCurrentToFirst();

}

private void DeleteCustomerCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

// If existing window is visible, delete the customer and all their orders.

// In a real application, you should add warnings and allow the user to cancel the operation.

var cur = custViewSource.View.CurrentItem as Customer;

var cust = (from c in context.Customers

where c.CustomerID == cur.CustomerID

select c).FirstOrDefault();

if (cust != null)

{

foreach (var ord in cust.Orders.ToList())

{

Delete\_Order(ord);

}

context.Customers.Remove(cust);

}

context.SaveChanges();

custViewSource.View.Refresh();

}

// Commit changes from the new customer form, the new order form,

// or edits made to the existing customer form.

private void UpdateCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

if (newCustomerGrid.IsVisible)

{

// Create a new object because the old one

// is being tracked by EF now.

Customer newCustomer = new Customer

{

Address = add\_addressTextBox.Text,

City = add\_cityTextBox.Text,

CompanyName = add\_companyNameTextBox.Text,

ContactName = add\_contactNameTextBox.Text,

ContactTitle = add\_contactTitleTextBox.Text,

Country = add\_countryTextBox.Text,

CustomerID = add\_customerIDTextBox.Text,

Fax = add\_faxTextBox.Text,

Phone = add\_phoneTextBox.Text,

PostalCode = add\_postalCodeTextBox.Text,

Region = add\_regionTextBox.Text

};

// Perform very basic validation

if (newCustomer.CustomerID.Length == 5)

{

// Insert the new customer at correct position:

int len = context.Customers.Local.Count();

int pos = len;

for (int i = 0; i < len; ++i)

{

if (String.CompareOrdinal(newCustomer.CustomerID, context.Customers.Local[i].CustomerID) < 0)

{

pos = i;

break;

}

}

context.Customers.Local.Insert(pos, newCustomer);

custViewSource.View.Refresh();

custViewSource.View.MoveCurrentTo(newCustomer);

}

else

{

MessageBox.Show("CustomerID must have 5 characters.");

}

newCustomerGrid.Visibility = Visibility.Collapsed;

existingCustomerGrid.Visibility = Visibility.Visible;

}

else if (newOrderGrid.IsVisible)

{

// Order ID is auto-generated so we don't set it here.

// For CustomerID, address, etc we use the values from current customer.

// User can modify these in the datagrid after the order is entered.

Customer currentCustomer = (Customer)custViewSource.View.CurrentItem;

Order newOrder = new Order()

{

OrderDate = add\_orderDatePicker.SelectedDate,

RequiredDate = add\_requiredDatePicker.SelectedDate,

ShippedDate = add\_shippedDatePicker.SelectedDate,

CustomerID = currentCustomer.CustomerID,

ShipAddress = currentCustomer.Address,

ShipCity = currentCustomer.City,

ShipCountry = currentCustomer.Country,

ShipName = currentCustomer.CompanyName,

ShipPostalCode = currentCustomer.PostalCode,

ShipRegion = currentCustomer.Region

};

try

{

newOrder.EmployeeID = Int32.Parse(add\_employeeIDTextBox.Text);

}

catch

{

MessageBox.Show("EmployeeID must be a valid integer value.");

return;

}

try

{

// Exercise for the reader if you are using Northwind:

// Add the Northwind Shippers table to the model.

// Acceptable ShipperID values are 1, 2, or 3.

if (add\_ShipViaTextBox.Text == "1" || add\_ShipViaTextBox.Text == "2"

|| add\_ShipViaTextBox.Text == "3")

{

newOrder.ShipVia = Convert.ToInt32(add\_ShipViaTextBox.Text);

}

else

{

MessageBox.Show("Shipper ID must be 1, 2, or 3 in Northwind.");

return;

}

}

catch

{

MessageBox.Show("Ship Via must be convertible to int");

return;

}

try

{

newOrder.Freight = Convert.ToDecimal(add\_freightTextBox.Text);

}

catch

{

MessageBox.Show("Freight must be convertible to decimal.");

return;

}

// Add the order into the EF model

context.Orders.Add(newOrder);

ordViewSource.View.Refresh();

}

// Save the changes, either for a new customer, a new order

// or an edit to an existing customer or order.

context.SaveChanges();

}

// Sets up the form so that user can enter data. Data is later

// saved when user clicks Commit.

private void AddCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

existingCustomerGrid.Visibility = Visibility.Collapsed;

newOrderGrid.Visibility = Visibility.Collapsed;

newCustomerGrid.Visibility = Visibility.Visible;

// Clear all the text boxes before adding a new customer.

foreach (var child in newCustomerGrid.Children)

{

var tb = child as TextBox;

if (tb != null)

{

tb.Text = "";

}

}

}

private void NewOrder\_click(object sender, RoutedEventArgs e)

{

var cust = custViewSource.View.CurrentItem as Customer;

if (cust == null)

{

MessageBox.Show("No customer selected.");

return;

}

existingCustomerGrid.Visibility = Visibility.Collapsed;

newCustomerGrid.Visibility = Visibility.Collapsed;

newOrderGrid.UpdateLayout();

newOrderGrid.Visibility = Visibility.Visible;

}

// Cancels any input into the new customer form

private void CancelCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

add\_addressTextBox.Text = "";

add\_cityTextBox.Text = "";

add\_companyNameTextBox.Text = "";

add\_contactNameTextBox.Text = "";

add\_contactTitleTextBox.Text = "";

add\_countryTextBox.Text = "";

add\_customerIDTextBox.Text = "";

add\_faxTextBox.Text = "";

add\_phoneTextBox.Text = "";

add\_postalCodeTextBox.Text = "";

add\_regionTextBox.Text = "";

existingCustomerGrid.Visibility = Visibility.Visible;

newCustomerGrid.Visibility = Visibility.Collapsed;

newOrderGrid.Visibility = Visibility.Collapsed;

}

private void Delete\_Order(Order order)

{

// Find the order in the EF model.

var ord = (from o in context.Orders.Local

where o.OrderID == order.OrderID

select o).FirstOrDefault();

// Delete all the order\_details that have

// this Order as a foreign key

foreach (var detail in ord.Order\_Details.ToList())

{

context.Order\_Details.Remove(detail);

}

// Now it's safe to delete the order.

context.Orders.Remove(ord);

context.SaveChanges();

// Update the data grid.

ordViewSource.View.Refresh();

}

private void DeleteOrderCommandHandler(object sender, ExecutedRoutedEventArgs e)

{

// Get the Order in the row in which the Delete button was clicked.

Order obj = e.Parameter as Order;

Delete\_Order(obj);

}

## Запуск приложения

Чтобы начать отладку, нажмите клавишу **F5**. Вы должны увидеть данные клиента и заказа, заполненные в сетке, и кнопки навигации должны работать должным образом. Щелкните **"Зафиксировать"** , чтобы добавить нового клиента или заказ в модель после ввода данных. Нажмите кнопку **"Отмена** ", чтобы вернуться из новой формы заказа или нового клиента, не сохраняя данные. Вы можете вносить изменения в существующие клиенты и заказы непосредственно в текстовых полях, а эти изменения автоматически записываются в модель.