



**Escuela Politécnica Superior de
Alicante**

Programación Avanzada de entornos de escritorio

Taller 7: Aplicaciones WPF: CRUD.

Contenidos

- Preparación del entorno
- Entity Framework
 - Conexión a la BD
- Listado
 - Detalle
 - Relaciones
 - Campos ampliados
- Añadir tuplas
- Eliminar tuplas
- Modificar tuplas

El entorno

- Necesitamos un servidor de base de datos accesible.
- Recomendamos:
 - SQL Server Express Edition en local.
 - SQL Server Management Studio
- Una vez arrancado el servicio de SQL Server conel Management Studio vamos a crear:
 - Una base de datos nueva: Taller7
 - Un login (o incicio de sesión nuevo): taller7 con su clave.
 - Daremos los permisos de propietario al login 'taller7' sobre la bd 'taller7'.

El entorno

- En la BD 'taller7' crearemos dos tablas:
 - Productos
 - Id, numérico, clave primaria
 - Nombre, varchar(200), not null
 - Descripción, varchar(2000), nullable
 - Precio, numérico con decimales, nullable
 - Categoría numérico, clave ajena → categorias.id
 - Categorias
 - Id: numérico, clave primaria
 - Nombre: varchar(100), not null

El entorno

- Ahora crearemos un nuevo proyecto WPF que llamaremos 'taller7'.
- Cambiamos el título de la ventana principal por 'Catálogo'.

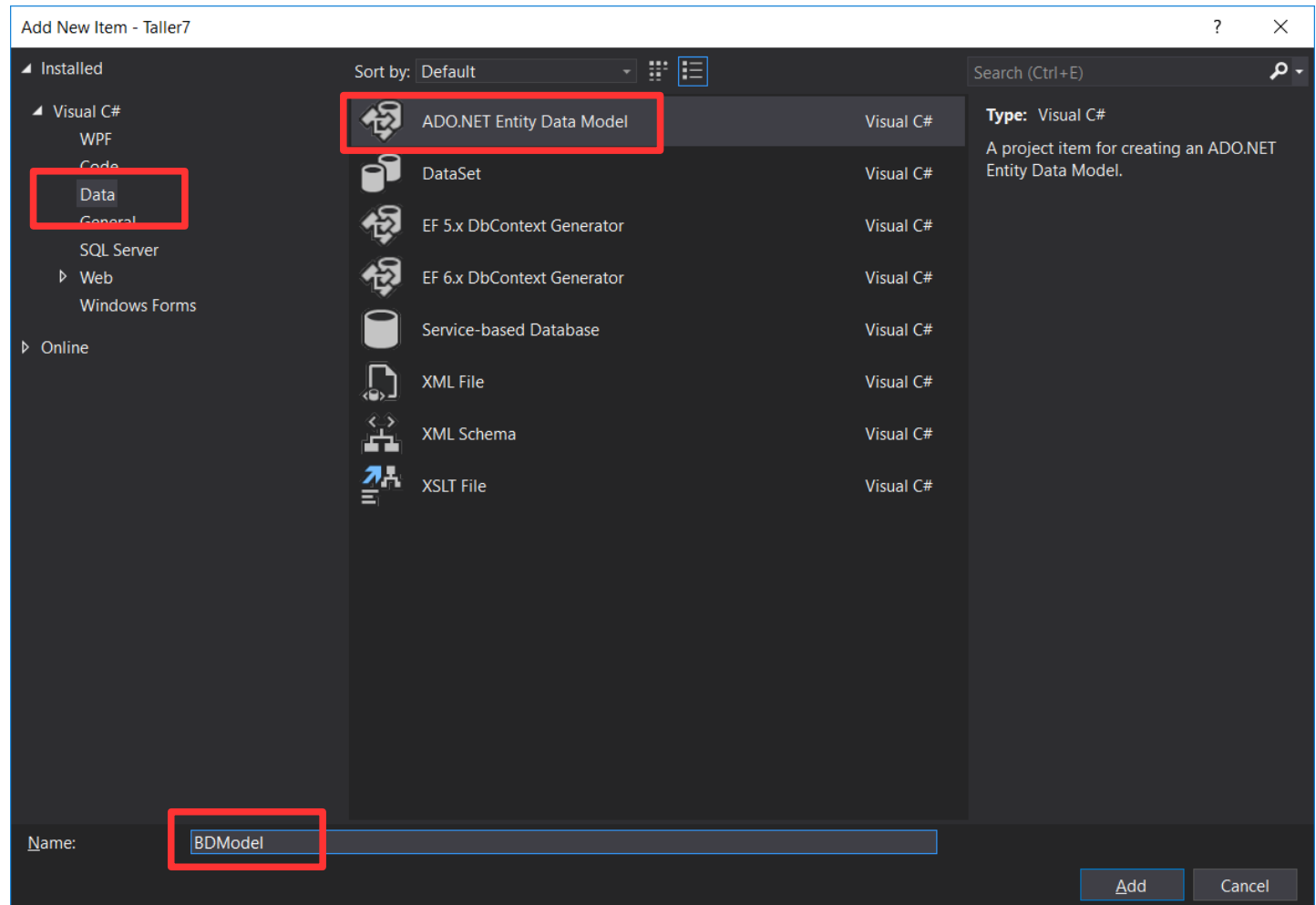
```
<Window x:Class="Taller7.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:Taller7"
  mc:Ignorable="d"
  Title="Catálogo" Height="450" Width="800">
  <Grid>
    ...
  </Grid>
</Window>
```

- Añadimos una etiqueta a modo de título:

```
<Label x:Name="lblTitulo" Content="Gestión del catálogo de
productos" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
```

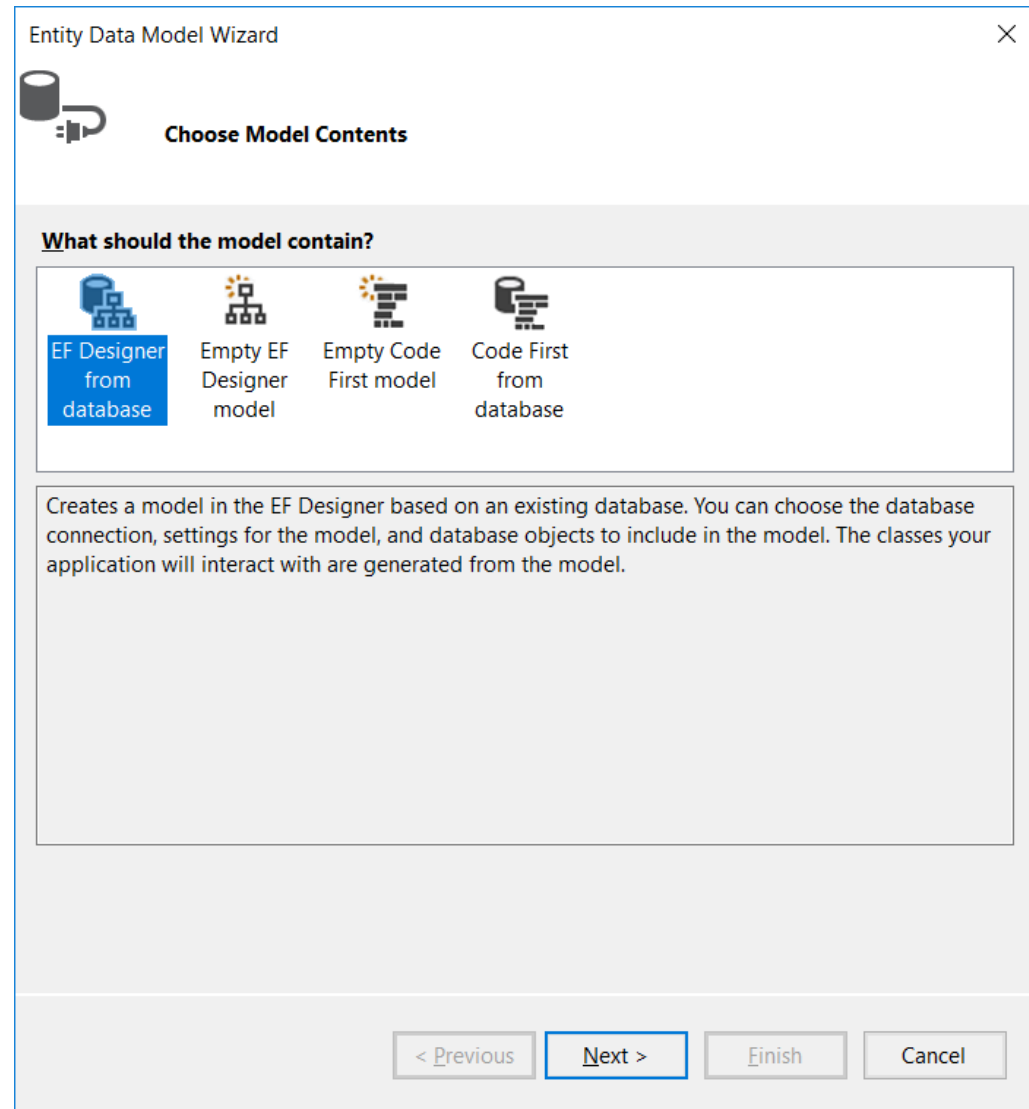
Crear el modelo EF

- Ahora añadiremos un modelo EF (Entity Framework)



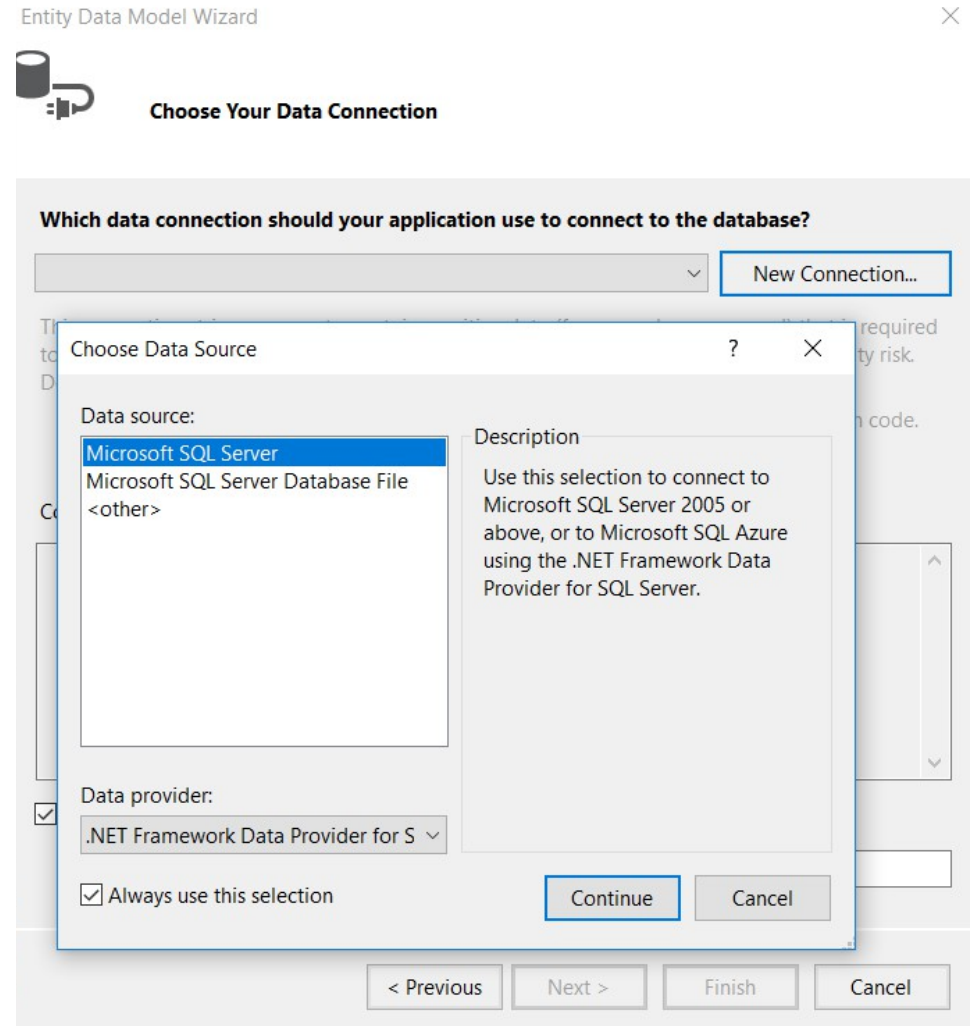
Crear el modelo EF

- Diseñador EF a partir de una base de datos existente



Crear el modelo EF

- Crearemos una nueva conexión en el proyecto, contra un servidor SQL Server.



Crear el modelo EF

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient) [Change...](#)

Server name: PORTATILJAUME\SQLEXPRESS [Refresh](#)

Log on to the server

Authentication: SQL Server Authentication

User name: taller7

Password: •••••

☐ Save my password

Connect to a database

☒ Select or enter a database name: taller7

☐ Attach a database file: [Browse...](#)

Logical name:

[Advanced...](#)

[Test Connection](#) [OK](#) [Cancel](#)

Entity Data Model Wizard

Choose Your Data Connection

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

portatiljaume\sqlexpress.taller7.taller7 [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:

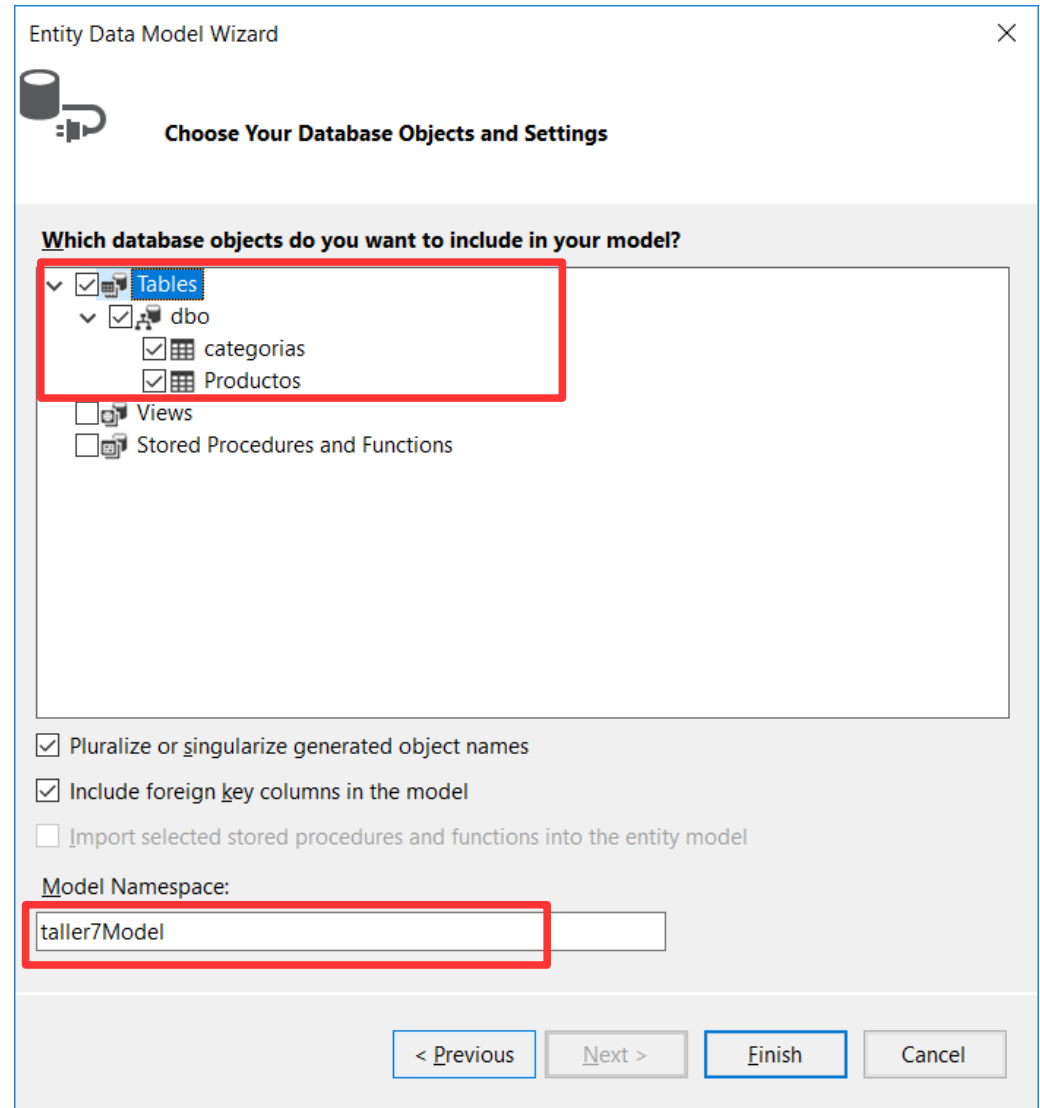
```
metadata=res://*/BDModel.csdl|res://*/BDModel.ssdl|
res://*/BDModel.msl;provider=System.Data.SqlClient;provider connection string="data
source=PORTATILJAUME\SQLEXPRESS;initial catalog=taller7;user
id=taller7;password=*****;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save connection settings in App.Config as: taller7Entities

[< Previous](#) [Next >](#) [Finish](#) [Cancel](#)

Crear el modelo EF

- Seleccionamos los objetos que deseamos incluir en el modelo: Las dos tablas creadas: categorias y productos.
- A tener en cuenta el 'Model namespace': taller7Model.



The image shows the 'Entity Data Model Wizard' dialog box, specifically the 'Choose Your Database Objects and Settings' step. The dialog has a title bar with 'Entity Data Model Wizard' and a close button. Below the title bar is a section with a database icon and the text 'Choose Your Database Objects and Settings'. The main area is titled 'Which database objects do you want to include in your model?'. It contains a tree view with the following items: 'Tables' (checked), 'dbo' (checked), 'categorias' (checked), 'Productos' (checked), 'Views' (unchecked), and 'Stored Procedures and Functions' (unchecked). A red rectangle highlights the 'Tables' section and its sub-items. Below the tree view are three checkboxes: 'Pluralize or singularize generated object names' (checked), 'Include foreign key columns in the model' (checked), and 'Import selected stored procedures and functions into the entity model' (unchecked). At the bottom, there is a 'Model Namespace:' label and a text box containing 'taller7Model', which is also highlighted with a red rectangle. At the very bottom are four buttons: '< Previous', 'Next >', 'Finish', 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
 - ☒ categorias
 - ☒ Productos
 - ☐ 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

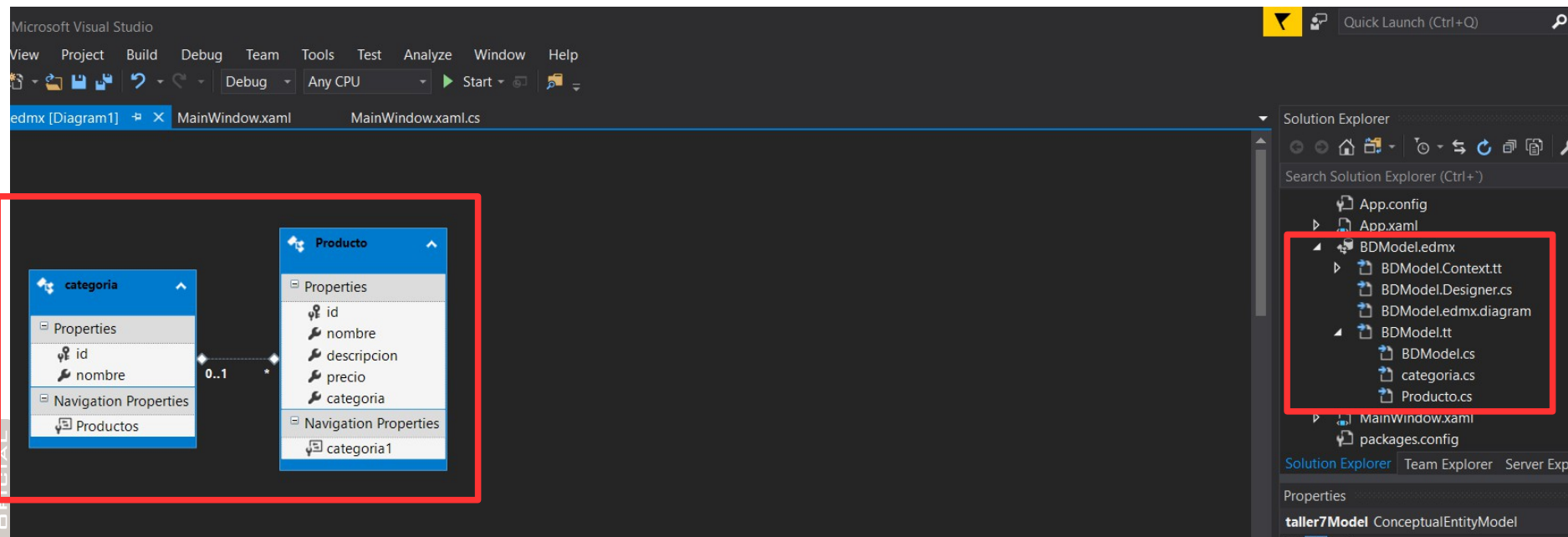
Model Namespace:

taller7Model

< Previous Next > Finish Cancel

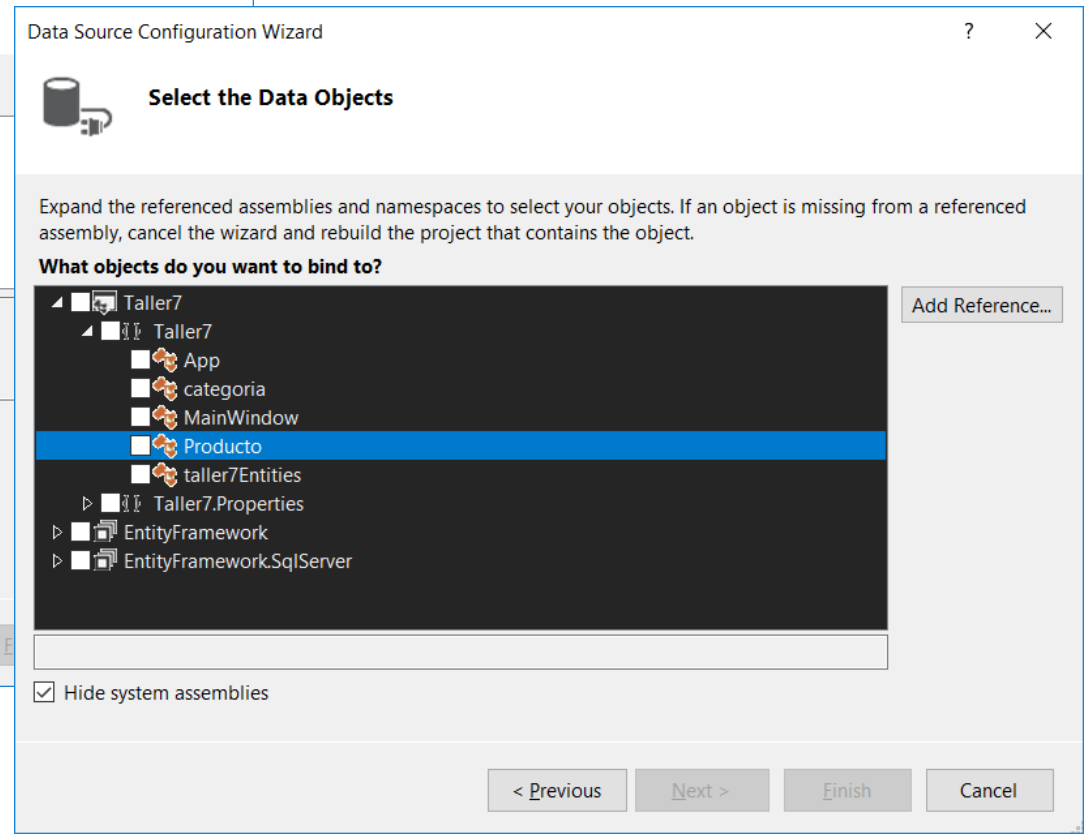
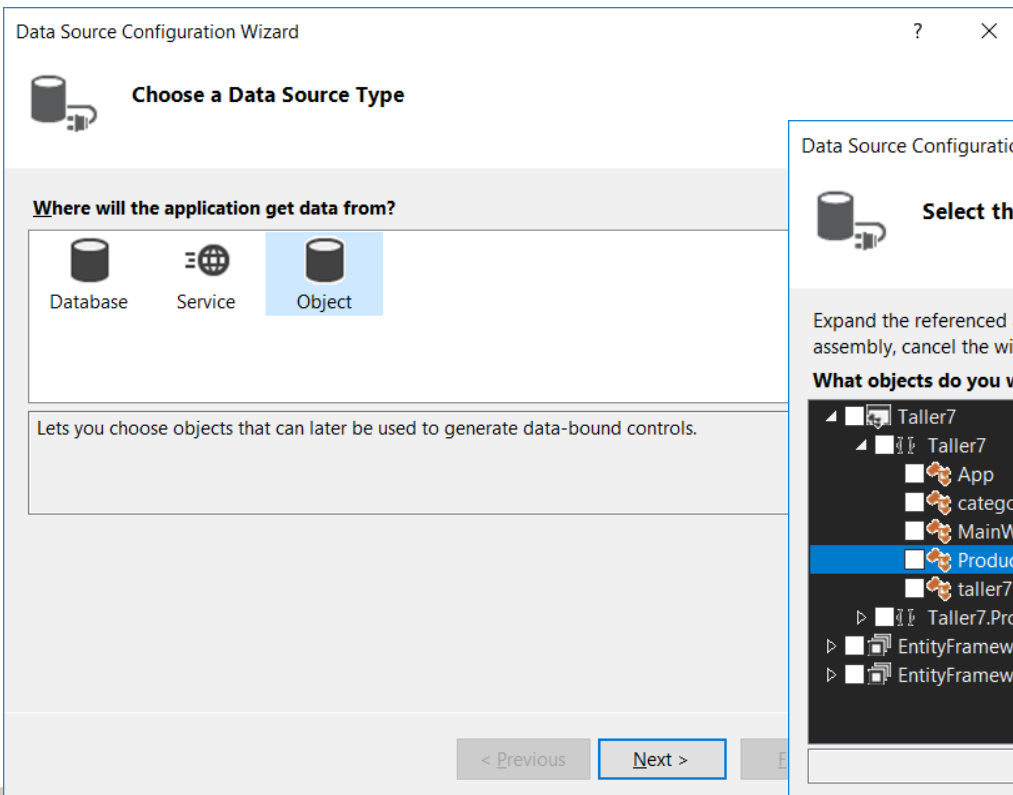
Crear el modelo EF

- Ahora tenemos las dos tablas vinculadas, como se puede observar en el diagrama del modelo EF.
- A la derecha podemos ver las clases y ficheros que ha generado la creación del modelo EF y la importación de los objetos desde la BD.



Listado básico con un dataGrid

- Primero crearemos un nuevo 'Data Source' (fuente de datos) de tipo 'objeto' contra la clase 'Producto'.



Listado básico con un dataGrid

- Ahora añadimos el data grid:
 - Desde la vista de diseño de 'MainWindow.xaml', seleccionamos de la barra lateral izquierda los 'Data sources'.
 - Desplegamos el combo box de productos y seleccionamos 'Datagrid' y a continuación arrastramos 'producto' y lo soltamos dentro de nuestro grid. Aparecerá un listado con las columnas de la tabla 'Productos'.
 - En el xaml, nos habrá creado un nuevo elemento de tipo 'DataGrid'.

Listado básico con un dataGrid

- Personalizamos el data grid:

```
<DataGrid x:Name="gridListado" AutoGenerateColumns="False"
EnableRowVirtualization="True" ItemsSource="{Binding}"
Margin="5,55,5,5" RowDetailsVisibilityMode="VisibleWhenSelected"
FontSize="14" FontFamily="Verdana" IsReadOnly="True"
AlternatingRowBackground="LightGray">

<DataGrid.Columns>

<DataGridTextColumn x:Name="idColumn" Binding="{Binding id}"
Header="Código" Width="Auto"/>

<DataGridTextColumn x:Name="nombreColumn" Binding="{Binding nombre}"
Header="Nombre" Width="200"/>

<DataGridTextColumn x:Name="descripcionColumn" Binding="{Binding
descripcion}" Header="Descripción" Width="*/>

<DataGridTextColumn x:Name="categoriaColumn" Binding="{Binding
categoria}" Header="Categoría" Width="100"/>

<DataGridTextColumn x:Name="precioColumn" Binding="{Binding precio}"
Header="Precio" Width="Auto"/>

</DataGrid.Columns></DataGrid>
```

Listado básico con un dataGrid

- Debemos indicar la fuente de datos del grid:

```
<Grid DataContext="{StaticResource productoViewSource}">
```

- Y en MainWindow.xaml.cs, debemos hacer la carga de datos:

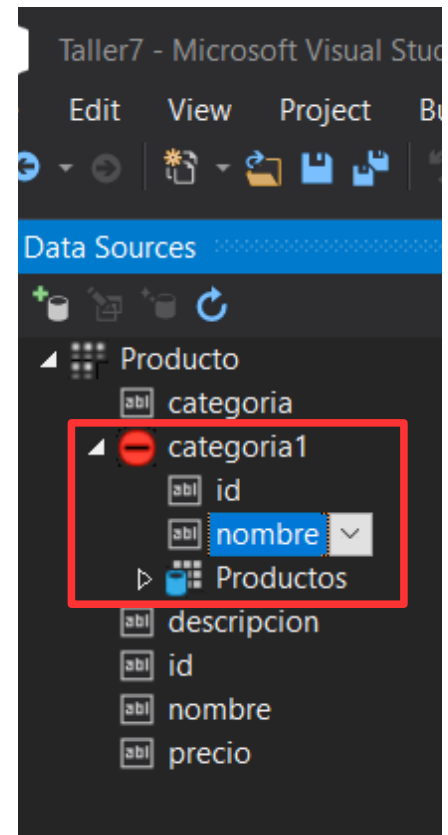
```
public partial class MainWindow : Window
{
    taller7Entities db;
    0 references
    public MainWindow()
    {
        InitializeComponent();
        db = new taller7Entities();
    }

    1 reference
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        System.Windows.Data.CollectionViewSource productoViewSource = ((System.Windows.Data.CollectionViewSource)(this.FindResource("productoViewSource")));
        // Load data by setting the CollectionViewSource.Source property:
        // productoViewSource.Source = [generic data source]
        db.Productos.Load();
        productoViewSource.Source = db.Productos.Local;
    }
}
```


Mejoras al listado simple

- Si queremos mostrar el nombre de la categoría en lugar de su id, tenemos que ver en el 'Data source' como se llama el objeto 'agregado' resultante de la relación y el nombre del atributo 'nombre'.
- Ahora vinculamos ésta propiedad en el 'binding' de la columna categoría:

```
<DataGridTextColumn  
  x:Name="categoriaColumn"  
  Binding="{Binding categoria1.nombre}"  
  Header="Categoría" Width="100"/>
```



Mejoras al listado simple

- Vamos a añadir en cada fila una botonera que permita realizar operaciones individuales sobre los datos.
- Quitaremos la columna con la descripción del producto y modificaremos los anchos de las demás celdas:

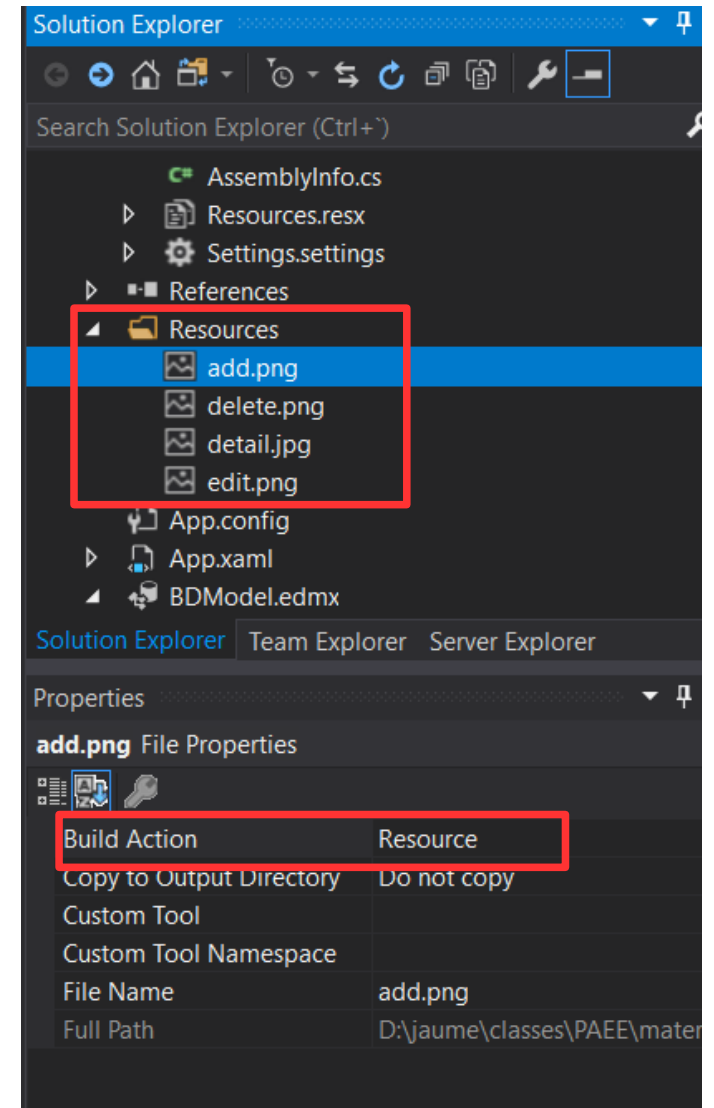
```
<DataGrid.Columns>
    <DataGridTextColumn x:Name="idColumn" Binding="{Binding id}"
Header="Código" Width="Auto"/>
    <DataGridTextColumn x:Name="nombreColumn" Binding="{Binding nombre}"
Header="Nombre" Width="390"/>
    <!--DataGridTextColumn x:Name="descripcionColumn" Binding="{Binding
descripcion}" Header="Descripción" Width="*" / -->
    <DataGridTextColumn x:Name="categoriaColumn" Binding="{Binding
categoria1.nombre}" Header="Categoría" Width="170"/>
    <DataGridTextColumn x:Name="precioColumn" Binding="{Binding precio}"
Header="Precio" Width="Auto"/>
```

Mejoras al listado simple

- A continuación añadiremos una nueva columna de tipo 'DataGridTemplateColumn', dentro de la cual definiremos un 'CellTemplate' y en el ubicaremos un 'DataTemplate'.
- Dentro de este último, intertaremos un StackPanel horizontal el cual contendrá tres botones:
 - Para la vista de detalle
 - Para editar el dato
 - Para eliminar la tupla
- Los botones los vincularemos a imágenes en lugar de etiquetas textuales.

Vincular una imagen

- Para vincular un fichero que contiene una imagen como un recurso de nuestro proyecto debemos:
 - Añadir una nueva carpeta llamada 'Resources'.
 - Copiar en la carpeta anterior las imagenes que deseamos vincular como recursos (incluyendolas en el proyecto)
 - Comprobar que en las propiedades de las imágenes, la propiedad 'Build Action' esta establecida a 'Resource'.



Mejoras al listado simple

- De esta forma nuestra Template Column quedará:

```
<DataGridTemplateColumn Header="" Width="Auto">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <Button x:Name="btnDetail" Background="LightGreen" Foreground="White" FontFamily="Verdana"
                    FontWeight="Bold" Margin="0, 0, 5, 0">
                    <StackPanel>
                        <Image Source="Resources/detail.jpg" Height="20" Width="20" />
                    </StackPanel>
                </Button>

                <Button x:Name="btnEdit" Background="LightYellow" Foreground="#000033" FontFamily="Verdana"
                    FontWeight="Bold" Margin="0, 0, 5, 0">
                    <StackPanel>
                        <Image Source="Resources/edit.png" Height="20" Width="20" />
                    </StackPanel>
                </Button>

                <Button x:Name="btnDelete" Background="Red" Foreground="White" FontFamily="Verdana"
                    FontWeight="Bold">
                    <StackPanel>
                        <Image Source="Resources/delete.png" Height="20" Width="20" />
                    </StackPanel>
                </Button>
            </StackPanel>
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
```

Vista detalle

- Vamos a añadir los objetos y código necesarios para implementar una vista detalle de un producto.
- Primero crearemos un nuevo grid, que ubicaremos a un lado de la ventana (fuera de ella), con las mismas dimensiones que el grid principal.
- Dentro de él pondremos un título del detalle y un botón de volver:

```
<Grid x:Name="gridCajaDetalle" Margin="825,55,-810,5"
Background="White">
    <Label x:Name="lblTituloDetalle" Content="Detalle del producto"
HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top"
FontSize="20" FontWeight="Bold"/>
    ...
    <Button Content="Volver" HorizontalAlignment="Left"
Margin="695,320,0,0" VerticalAlignment="Top"
    Background="Red" Foreground="#000033" Width="74" Height="30"
FontFamily="Verdana" FontSize="12" FontWeight="Bold"
Click="Button_Click"/>
</Grid>
```

Vista detalle

- Ahora añadimos un nuevo data grid:
 - Seleccionamos de la barra lateral izquierda los 'Data sources'.
 - Desplegamos el combo box de productos y seleccionamos 'Details' (detalles) y a continuación arrastramos 'producto' y lo soltamos dentro del nuevo grid 'gridCajaDetalle'. Aparecerá un formulario a dos columnas con los campos de la tabla 'Productos'.
 - En el xaml, nos habrá creado un nuevo elemento de tipo 'Grid' que renombraremos como 'gridDetalle'

Vista detalle

- Modificamos el nuevo grid para adecuar su aspecto:
 - Cambiaremos altura y anchura de algunas de sus files/columnas
 - Para los campos:
 - Cambiaremos la forma del binding
 - Los haremos de solo lectura
 - Modificaremos su orden
 - Decoraremos con tipografía y colores diferentes.

Vista detalle

- Modificamos el nuevo grid para adecuar su aspecto:

```
<Grid x:Name="gridCajaDetalle" Margin="825,55,-810,5" Background="White">
    <Label x:Name="lblTituloDetalle" Content="Detalle del producto" HorizontalAlignment="Left"
        Margin="10,10,0,0" VerticalAlignment="Top" FontSize="20" FontWeight="Bold"/>
    <Grid x:Name="gridDetalle" HorizontalAlignment="Left" VerticalAlignment="Top" Height="263"
        Margin="-1,52,0,0" Width="770">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="2"/>
            <ColumnDefinition Width="100"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="75"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
    </Grid>
</Grid>
```

Vista detalle

- Modificamos el nuevo grid para adecuar su aspecto, continuación:

```
<Label Content="Código:" Grid.Column="1" HorizontalAlignment="Left" Margin="0.6,3,0,3" Grid.Row="0"
VerticalAlignment="Center" FontSize="14" FontWeight="Bold"/>
<TextBox IsReadOnly="True" x:Name="idTextBox" Grid.Column="2" HorizontalAlignment="Left" Height="24"
Margin="3,4,0,4" Grid.Row="0" Text="{Binding id}" VerticalAlignment="Center" Width="120" FontSize="16"/>

<Label Content="Nombre:" Grid.Column="1" HorizontalAlignment="Left" Margin="3,3,0,3" Grid.Row="1"
VerticalAlignment="Center" FontSize="14" FontWeight="Bold"/>
<TextBox IsReadOnly="True" x:Name="nombreTextBox" Grid.Column="2" HorizontalAlignment="Left" Height="24"
Margin="3.8,6,-524.8,6.8" Grid.Row="1" Text="{Binding nombre}" VerticalAlignment="Center" Width="653" FontSize="16"/>

<Label Content="Descripción:" HorizontalAlignment="Left" Margin="3,3,0,36.8" Grid.Row="2"
VerticalAlignment="Center" FontSize="14" FontWeight="Bold" Grid.ColumnSpan="2"/>
<TextBox IsReadOnly="True" x:Name="descripcionTextBox" Grid.Column="2" HorizontalAlignment="Left" Height="60"
Margin="2.8,6.2,-525.6,9.8" Grid.Row="2" Text="{Binding descripcion}" VerticalAlignment="Center" Width="654"
FontSize="16"/>

<Label Content="Categoría:" Grid.Column="1" HorizontalAlignment="Left" Margin="3,3,0,3" Grid.Row="3"
VerticalAlignment="Center" FontSize="14" FontWeight="Bold"/>
<TextBox IsReadOnly="True" x:Name="categoriaTextBox" Grid.Column="2" HorizontalAlignment="Left" Height="24"
Margin="2.8,6.4,-253.2,7.2" Grid.Row="3" Text="{Binding categoria1.nombre}" VerticalAlignment="Center" Width="384"
FontSize="16"/>

<Label Content="Precio:" Grid.Column="1" HorizontalAlignment="Left" Margin="3,3,0,3" Grid.Row="4"
VerticalAlignment="Center" FontSize="14" FontWeight="Bold"/>
<TextBox IsReadOnly="True" x:Name="precioTextBox" Grid.Column="2" HorizontalAlignment="Left" Height="24"
Margin="2.8,4,0,4" Grid.Row="4" Text="{Binding precio}" VerticalAlignment="Center" Width="120" FontSize="16"/>
<Label Content="euros" Grid.Column="2" HorizontalAlignment="Left" Margin="125,4,-54.6,9" Grid.Row="4"
VerticalAlignment="Center" FontSize="14" FontWeight="Bold"/>
</Grid>
```

Vista detalle

- Añadimos el evento click al botón de detalle:

```
<Button x:Name="btnDetail" Background="LightGreen"
Foreground="White" FontFamily="Verdana"
FontWeight="Bold" Margin="0, 0, 5, 0" Click="btnDetail_Click" >
<StackPanel>
<Image Source="Resources/detail.jpg" Height="20" Width="20" />
</StackPanel>
</Button>
```

Vista detalle

- Finalmente, tenemos que añadir código para los botones de detalle y de volver:

```
public void btnDetail_Click(object sender, EventArgs e) {  
    Producto p = (Producto)gridListado.SelectedItem;  
    lblTituloDetalle.Content = "Detalle del producto: '" +  
p.nombre + "'";  
    gridListado.Visibility = Visibility.Hidden;  
    gridCajaDetalle.Margin = new Thickness(5, 55, 5, 4);  
    gridCajaDetalle.Visibility = Visibility.Visible;  
}  
  
private void btnVolverDetalle_Click(object sender,  
RoutedEventArgs e) {  
    gridCajaDetalle.Margin = new Thickness(825, 55, -650, 5);  
    gridCajaDetalle.Visibility = Visibility.Hidden;  
    gridListado.Visibility = Visibility.Visible;  
}
```

Añadir productos

- Ahora vamos a implementar la opción de crear un nuevo producto:
 - Primero incluimos un botón al lado del título del listado que nos dé acceso a esta opción:

```
<Button x:Name="btnNuevo" HorizontalAlignment="Left"
Margin="693,10,0,0" VerticalAlignment="Top" Width="91"
Height="40" Background="LightBlue" Foreground="#000033">
    <StackPanel Width="77" Height="30" Orientation="Horizontal">
        <Image Source="Resources/add.png" Height="20"></Image>
        <TextBlock VerticalAlignment="Center" FontFamily="Verdana"
        FontWeight="Bold" FontSize="14">Añadir</TextBlock>
    </StackPanel>
</Button>
```

Añadir productos

- Ahora configuramos el entorno, añadiendo un nuevo grid y dentro de él, un título y un botón para volver al listado.

```
<Grid x:Name="gridCajaNuevo" Margin="5,-393,3.6,453"
Background="White">
    <Label x:Name="lblTituloNuevo" Content="Añadir un nuevo
producto" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" FontSize="20" FontWeight="Bold"/>
    <Button x:Name="btnVolverNuevo" Content="Volver"
HorizontalAlignment="Left" Margin="695,320,0,0"
VerticalAlignment="Top" Background="Red" Foreground="#000033"
Width="74" Height="30" FontFamily="Verdana" FontSize="12"
FontWeight="Bold" Click="btnVolverNuevo_Click"/>
</Grid>
```

Añadir productos

- Ahora, el código de los botones btnNuevo y btnVolverNuevo_click:

```
private void btnNuevo_Click(object sender, RoutedEventArgs e) {  
    gridListado.Visibility = Visibility.Hidden;  
    btnNuevo.Visibility = Visibility.Hidden;  
    gridCajaNuevo.Margin = new Thickness(5, 55, 5, 4);  
    gridCajaNuevo.Visibility = Visibility.Visible;  
}  
  
private void btnVolverNuevo_Click (object sender, RoutedEventArgs  
e) {  
    gridCajaNuevo.Margin = new Thickness(5, -393, 3.6, 453);  
    gridCajaNuevo.Visibility = Visibility.Hidden;  
    gridListado.Visibility = Visibility.Visible;  
    btnNuevo.Visibility = Visibility.Visible;  
}
```

Añadir productos

- El siguiente paso sera insertar dentro del nuevo grid otro grid de tipo 'details', tal y como hicimos con la opción de vista detalle, para después personalizarlo.
 - Abrir los 'Data Sources', seleccionar plantilla 'details' y arrastrar dentro del grid 'gridCajaNuevo'
- Eliminaremos los bindings, ocultaremos el código, cambiaremos el aspecto y el orden de los campos.
- Finalmente, cambiaremos el 'textbox' de la categoría por un combobox (desplegable) que muestre todas las categorías disponibles.

Desplegable con las categorías

- Añadimos un comboBox en sustitución del TextBox de categorías:

```
<ComboBox x:Name="slctCategorias" Grid.Column="2"  
HorizontalAlignment="Left" Height="28"  
Background="White" Margin="2.8,6.4,-253.2,7.2"  
Grid.Row="3" VerticalAlignment="Center" Width="300"  
FontSize="16">  
  
</ComboBox>
```

Desplegable con las categorías

- Creamos una función para 'poblar' el desplegable desde la base de datos:

```
private void rellenarComboCategorias()  
{  
    List<categoria> cats = db.categorias.OrderBy(c =>  
c.nombre).ToList();  
    slctCategorias.ItemsSource = cats;  
    slctCategorias.DisplayMemberPath = "nombre";  
    slctCategorias.SelectedValuePath = "id";  
}
```

Desplegable con las categorías

- Y la invocamos al acceder al formulario del nuevo producto:

```
private void btnNuevo_Click(object sender,
RoutedEventArgs e) {
    LimpiarFormNuevo();
    gridListado.Visibility = Visibility.Hidden;
    btnNuevo.Visibility = Visibility.Hidden;
    gridCajaNuevo.Margin = new Thickness(5, 55, 5, 4);
    gridCajaNuevo.Visibility = Visibility.Visible;
    rellenarComboCategorias();
}
```

Desplegable con las categorías

- Esta función vacía el formulario de nuevo producto de anteriores usos:

```
private void limpiarFormNuevo()  
{  
    nombreTextBoxNuevo.Text = "";  
    descripcionTextBoxNuevo.Text = "";  
    slctCategorias.SelectedIndex = 0;  
    precioTextBoxNuevo.Text = "";  
}
```

Botón de añadir el dato

- Añadiremos un botón para realizar la inserción del nuevo producto en la base de datos. Lo ubicaremos al lado del botón de volver al listado:

```
<Button x:Name="btnAddNuevo" Content="Añadir"  
HorizontalAlignment="Left" Margin="600,320,0,0"  
VerticalAlignment="Top" Background="LightBlue"  
Foreground="#000033" Width="75" Height="30"  
FontFamily="Verdana" FontSize="12" FontWeight="Bold"  
Click="btnAddNuevo_Click"/>
```

Botón de añadir el dato

- El evento del botón 'btnAddNuevo' debe:
 - Verificar que los datos son correctos
 - Crear un nuevo objeto 'Producto' y dar valor a sus atributos desde los controles del formulario.
 - Enviar el nuevo dato a la base de datos.
 - Capturar algún posible error durante la transacción e informar al usuario.
 - Si todo ha ido bien, informar al usuario y volver al listado.

Botón de añadir el dato

- El código del evento del botón 'btnAddNuevo':

```
private void btnAddNuevo_Click(object sender, RoutedEventArgs e)
{
    //TODO: verificar que los datos del formulario son correctos
    Producto nuevo = new Producto();
    try
    {
        nuevo.Id = Convert.ToInt32(idTextBoxNuevo.Text);
        nuevo.Nombre = nombreTextBoxNuevo.Text;
        nuevo.Descripción = descripcionTextBoxNuevo.Text;
        nuevo.Precio = Convert.ToDecimal(precioTextBoxNuevo.Text);
        nuevo.Categoría = Convert.ToInt32(slctCategorias.SelectedValue.ToString());

        db.Productos.Add(nuevo);
        db.SaveChanges();

        gridCajaNuevo.Margin = new Thickness(5, -393, 3.6, 453);
        gridCajaNuevo.Visibility = Visibility.Hidden;
        gridListado.Visibility = Visibility.Visible;
        btnNuevo.Visibility = Visibility.Visible;

        MessageBox.Show("Producto '" + nuevo.nombre + "' añadido correctamente.",
            "Atención!", MessageBoxButton.OK, MessageBoxImage.Information);
    } catch (Exception ex)
    {
        MessageBox.Show("Error al añadir un nuevo producto. Causa:" + ex.Message,
            "Atención!", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Esto es igual que el evento 'btnVolverNuevo_Click', por lo que sería conveniente encapsularlo en una función e invocarlo desde aquí y desde el anterior evento.

Eliminar productos

- Para eliminar un producto:
 - Asignamos un manejador del evento de click sobre los botones de borrado:

```
<Button x:Name="btnDelete" Background="Red"  
Foreground="White" FontFamily="Verdana" FontWeight="Bold"  
Click="btnDelete_Click">  
...  
</Button>
```

- Este manejador, pedirá confirmación al usuario
- En caso afirmativo, se eliminará el producto y se informará del resultado.
- En caso de error, capturaremos la excepción e informaremos convenientemente.

Eliminar productos

- El código del manejador del evento de click sobre los botones de borrado:

```
private void btnDelete_Click(object sender, EventArgs e)
{
    Producto p = (Producto)gridListado.SelectedItem;
    MessageBoxResult res = MessageBox.Show("¿Está seguro que desea borrar el producto '" + p.nombre + "'"
        , "Atención!", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (res==MessageBoxResult.Yes)
    {
        try
        {
            db.Productos.Remove(p);
            db.SaveChanges();

            MessageBox.Show("Producto '" + p.nombre + "'" + " eliminado correctamente.",
                "Atención!", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al eliminar un producto. Causa:" + ex.Message,
                "Atención!", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

Editar productos

- Nos queda la última operación del CRUD: modificar un producto.
- Para ello vamos a crear el contexto necesario:
 - Añadimos un nuevo grid: gridCajaEditar, mismas dimensiones que el gridCajaNuevo, y lo ubicamos abajo del listado.
 - Dentro del anterior grid ubicaremos una etiqueta para el título (lblTituloEditar), un botón para volver al listado (btnVolverEdit) y otro para aplicar las modificaciones en la BD (btnEditProd).

Editar productos

- Código XAML del nuevo grid:

```
<Grid x:Name="gridCajaEditar" Margin="5,432,3.6,-372"
Background="White">
    <Label x:Name="lblTituloEditar" Content="Modificación de un
    producto" HorizontalAlignment="Left" Margin="10,10,0,0"
    VerticalAlignment="Top" FontSize="20" FontWeight="Bold"/>
    <!-- aquí irá el formulario -->
    <Button x:Name="btnEditProd" Content="Modificar"
    HorizontalAlignment="Left" Margin="600,320,0,0"
    VerticalAlignment="Top" Background="LightBlue"
    Foreground="#000033" Width="75" Height="30" FontFamily="Verdana"
    FontSize="12" FontWeight="Bold" Click="btnEditProd_Click"/>

    <Button x:Name="btnVolverEdit" Content="Volver"
    HorizontalAlignment="Left" Margin="695,320,0,0"
    VerticalAlignment="Top" Background="Red" Foreground="#000033"
    Width="74" Height="30" FontFamily="Verdana" FontSize="12"
    FontWeight="Bold" Click="btnVolverEdit_Click"/>
</Grid>
```

Editar productos

- Ahora, vamos a crear el formulario a partir de la plantilla 'details' del data source 'Producto'. De la misma forma que hicimos con la opción 'Detalle' y 'Añadir'.
- Abrimos el panel de 'Data sources', seleccionamos 'Details' del desplegable de Productos y finalmente arrastramos y soltamos dentro del nuevo grid.
- Después modificaremos el código del grid generado generado para adaptarlo a nuestro aplicativo: cambiando tamaños, estilos y otras cuestiones.
 - Eliminamos el campo 'Código'

Sustituimos el TextBox de categoria por un
ComboBox: slctCategoriasEdit.

Editar productos

- Código del grid del formulario de modificar productos:

```
<Grid x:Name="gridEditor" HorizontalAlignment="Left"
VerticalAlignment="Top" Height="263"
Margin="-1,52,0,0" Width="770">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="2"/>
    <ColumnDefinition Width="100"/>
    <ColumnDefinition Width="Auto"/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="75"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
  </Grid.RowDefinitions>
```

Editar productos

- Código del grid del formulario de modificar productos:

```
<Label Content="Nombre:" Grid.Column="1" HorizontalAlignment="Left"
Margin="3,3,0,3" Grid.Row="1" VerticalAlignment="Center"
FontSize="14" FontWeight="Bold"/>
```

```
    <TextBox x:Name="nombreTextBoxEdit" Grid.Column="2"
HorizontalAlignment="Left" Height="24" Text="{Binding nombre,
Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}" Margin="3.8,6,-524.8,6.8" Grid.Row="1"
VerticalAlignment="Center" Width="653" FontSize="16"/>
```

```
    <Label Content="Descripción:" HorizontalAlignment="Left"
Margin="3,3,0,36.8" Grid.Row="2" VerticalAlignment="Center"
FontSize="14" FontWeight="Bold" Grid.ColumnSpan="2"/>
```

```
    <TextBox x:Name="descripcionTextBoxEdit" Grid.Column="2"
HorizontalAlignment="Left" Height="60" Margin="2.8,6.2,-525.6,9.8"
Grid.Row="2" VerticalAlignment="Center" Width="654"
```

```
        Text="{Binding descripcion, Mode=TwoWay,
NotifyOnValidationError=true, ValidatesOnExceptions=true}"
FontSize="16"/>
```

Editar productos

- Código del grid del formulario de modificar productos:

```
<Label Content="Categoría:" Grid.Column="1" HorizontalAlignment="Left"
Margin="3,3,0,3" Grid.Row="3" VerticalAlignment="Center" FontSize="14"
FontWeight="Bold"/>
<ComboBox x:Name="slctCategoriasEdit" Grid.Column="2"
HorizontalAlignment="Left" Height="28" Background="White"
SelectedValue="{Binding categoria, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}" Margin="2.8,6.4,-253.2,7.2" Grid.Row="3"
VerticalAlignment="Center" Width="300" FontSize="16"/>
<Label Content="Precio:" Grid.Column="1" HorizontalAlignment="Left"
Margin="3,3,0,3" Grid.Row="4" VerticalAlignment="Center" FontSize="14"
FontWeight="Bold"/>
<TextBox x:Name="precioTextBoxEdit" Grid.Column="2" HorizontalAlignment="Left"
Height="24" Text="{Binding precio, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}" Margin="2.8,4,0,4" Grid.Row="4"
VerticalAlignment="Center" Width="120" FontSize="16"/>
<Label Content="euros" Grid.Column="2" HorizontalAlignment="Left"
Margin="125,4,-54.6,9" Grid.Row="4" VerticalAlignment="Center" FontSize="14"
FontWeight="Bold"/>
</Grid>
```

Editar productos

- Código del manejador del botón de volver desde el formulario de modificar productos:

```
private void btnVolverEdit_Click(object sender, RoutedEventArgs e) {  
    volverDesdeEdit();  
}
```

```
private void volverDesdeEdit() {  
    gridCajaEditar.Margin = new Thickness(5, 432, 3.6, -372);  
    gridCajaEditar.Visibility = Visibility.Hidden;  
    gridListado.Visibility = Visibility.Visible;  
    btnNuevo.Visibility = Visibility.Visible;  
}
```


Editar productos

- Tenemos que vincular el botón de editar del listado: btnEdit.

```
<Button x:Name="btnEdit" Background="LightYellow"
Foreground="#000033" FontFamily="Verdana"
    FontWeight="Bold" Margin="0, 0, 5, 0" Click="btnEdit_Click">
    ...
</Button>
```

- Este es el código del manejador:

```
private void btnEdit_Click(object sender, RoutedEventArgs e) {
    gridListado.Visibility = Visibility.Hidden;
    btnNuevo.Visibility = Visibility.Hidden;
    gridCajaEditar.Margin = new Thickness(5, 55, 5, 4);
    gridCajaEditar.Visibility = Visibility.Visible;
    rellenarComboCategorias(slctCategoriasEdit);
}
```

Editar productos

- Hemos cambiado la función que rellena las opciones del combo de categorías para poder reutilizarla, pasándole un control comboBox como argumento:

```
private void rellenarComboCategorias(ComboBox slct)
{
    List<categoria> cats = db.categorias
        .OrderBy(c => c.nombre).ToList();

    slct.ItemsSource = cats;
    slct.DisplayMemberPath = "nombre";
    slct.SelectedValuePath = "id";
}
```

Editar productos

- El manejador del botón que almacena la modificación en la base de datos:

```
private void btnEditProd_Click(object sender, RoutedEventArgs e) {  
    //TODO: verificar que los datos del formulario son correctos  
    Producto p = (Producto)gridListado.SelectedItem;  
    try {  
        db.SaveChanges();  
        volverDesdeEdit();  
        MessageBox.Show("Producto '" + p.nombre + "' modificado  
correctamente.", "Atención!", MessageBoxButton.OK,  
MessageBoxImage.Information);  
    } catch (Exception ex) {  
        MessageBox.Show("Error al modificar un producto. Causa:" +  
ex.Message, "Atención!", MessageBoxButton.OK, MessageBoxImage.Error);  
    }  
}
```

Ejercicios de ampliación, sencillos

- Ampliar el aplicativo anterior con el CRUD de productos añadiendo las siguientes prestaciones:
 - Verificar que los datos antes de añadir/modificar un producto.
 - Verificar en tiempo real que los datos de los formularios de añadir/modificar un producto son correctos y en caso contrario señalar los controles erróneos y deshabilitar el botón de añadir/modificar.
 - Cuando haya muchos productos en el listado, gestionar un scroll vertical que solo afecte al grid del listado.
 - Añadir un nuevo campo a la tabla productos:
 - Especificaciones: texto grande
 - Fecha de alta: usando un date picker
 - Subcategoría: con un segundo combo basado en una tabla subcategorias que se relaciona con categorías.

Ejercicios de ampliación, menos sencillos

- Ampliar el aplicativo anterior con el CRUD de productos añadiendo las siguientes prestaciones:
 - Gestionar una paginación en el listado
 - Añadir al listado un campo de texto que permita buscar/filtrar por nombre del producto.
 - En los formularios de añadir/editar un producto, permitir añadir/eliminar nuevas categorías mediante una ventana emergente que permita gestionarlas.

Referencias

- Algunos tutoriales o documentos on line que pueden servir para tener un enfoque diferente a lo explicado en este taller:
 - <https://docs.microsoft.com/en-us/visualstudio/data-tools/create-a-simple-data-application-with-wpf-and-entity-framework-6?view=vs-2019>
 - <https://www.c-sharpcorner.com/UploadFile/maresh/mastering-wpf-datagrid-in-a-day-hour-7-data-template/>
 - <https://parallelcodes.com/wpf-bind-combobox-sql-database/>