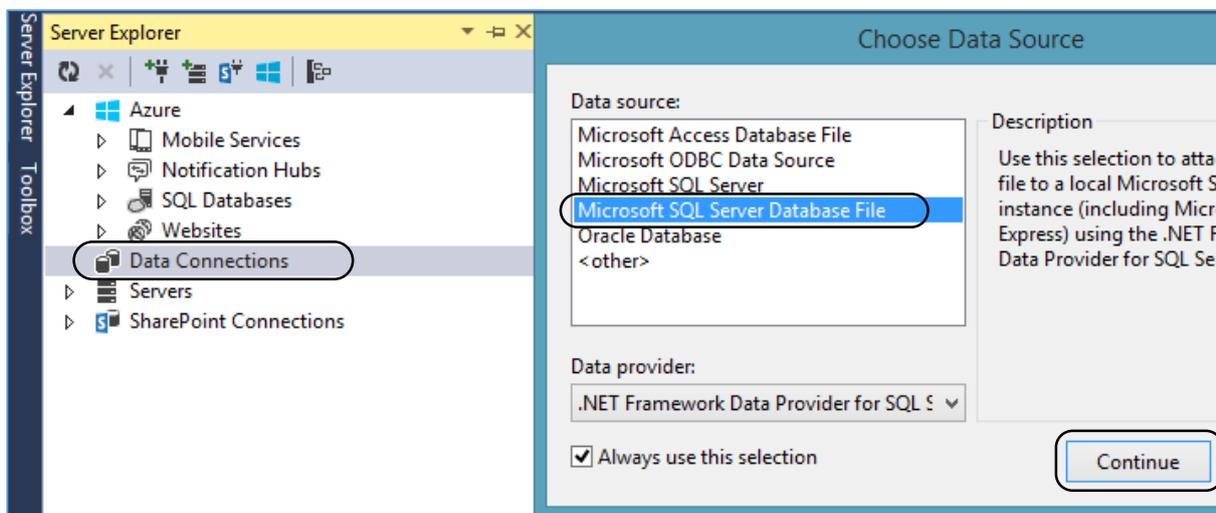


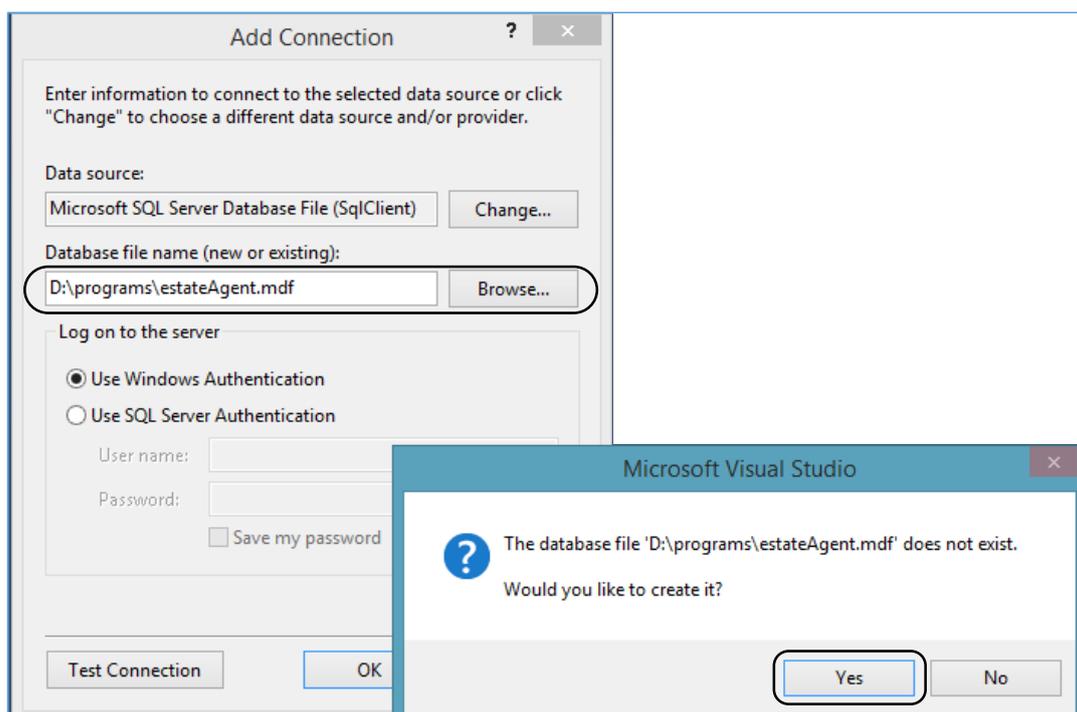
7 Estate Agent Database

The next program is a complete database application for an Estate Agent, handling properties for sale and customers wishing to find suitable properties. We will look at how C# .NET can **display existing records**, **add new records** to the database, **update records** and **delete records**, as well as carrying out **queries** to find suitable properties for particular customers.

Go to the **Server Explorer** window and right-click on **Data Connection** to set up a new connection. Select **Microsoft SQL Server Database File** as the Data source.

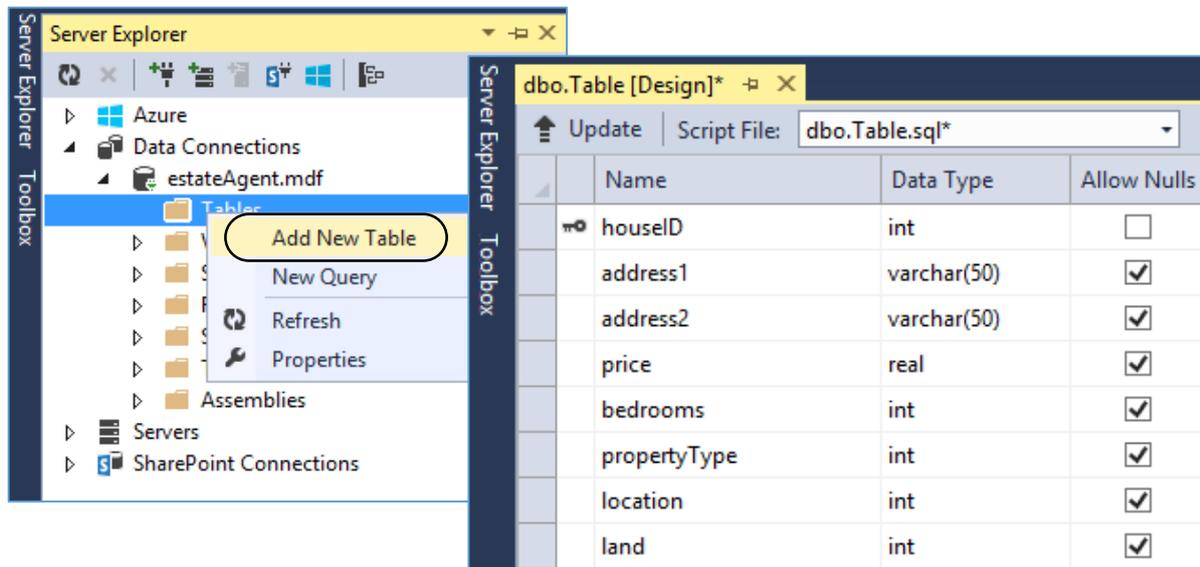


Use the Browse button to select a drive and folder location, then create a database called **'estateAgent'**.



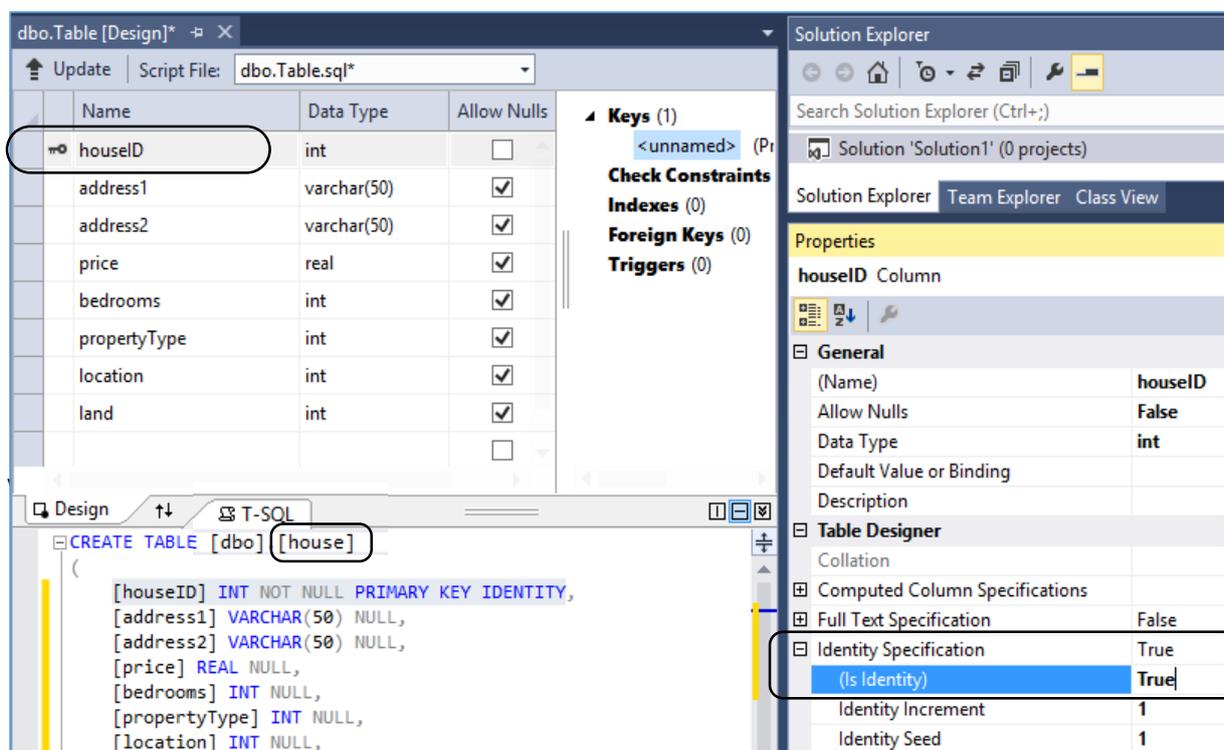
Right-click the **Tables** icon to add a new table to the database. This table will store details of houses for sale.

Add fields to the table as shown. The fields '**propertyType**', '**location**' and '**land**' will be code numbers representing different property descriptions.



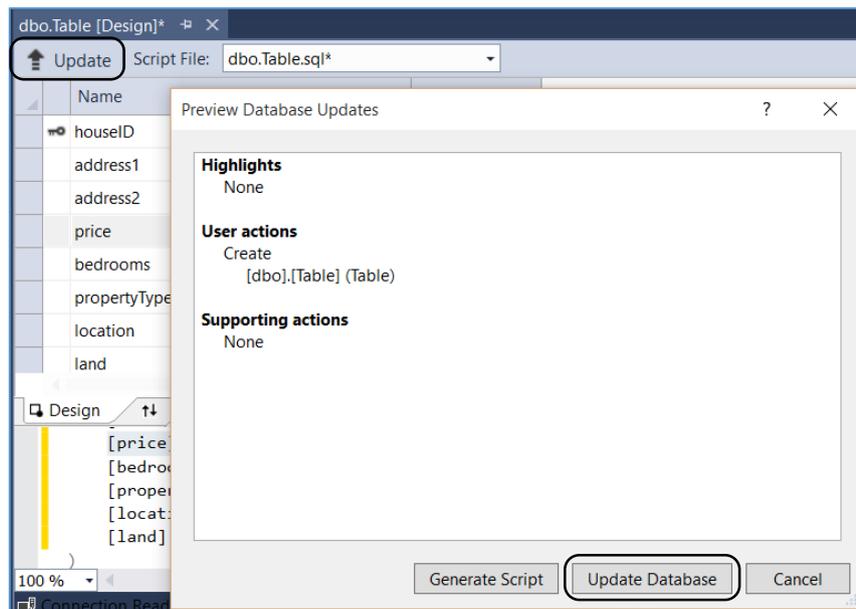
We will set the '**houseID**' to be an auto-number generated by the database.

Select the '**houseID**' field. Go to the Properties window and locate '**Identity Specification**'. Click the 'plus' symbol to the left to open further options, then set '**(Is Identity)**' to '**True**'.



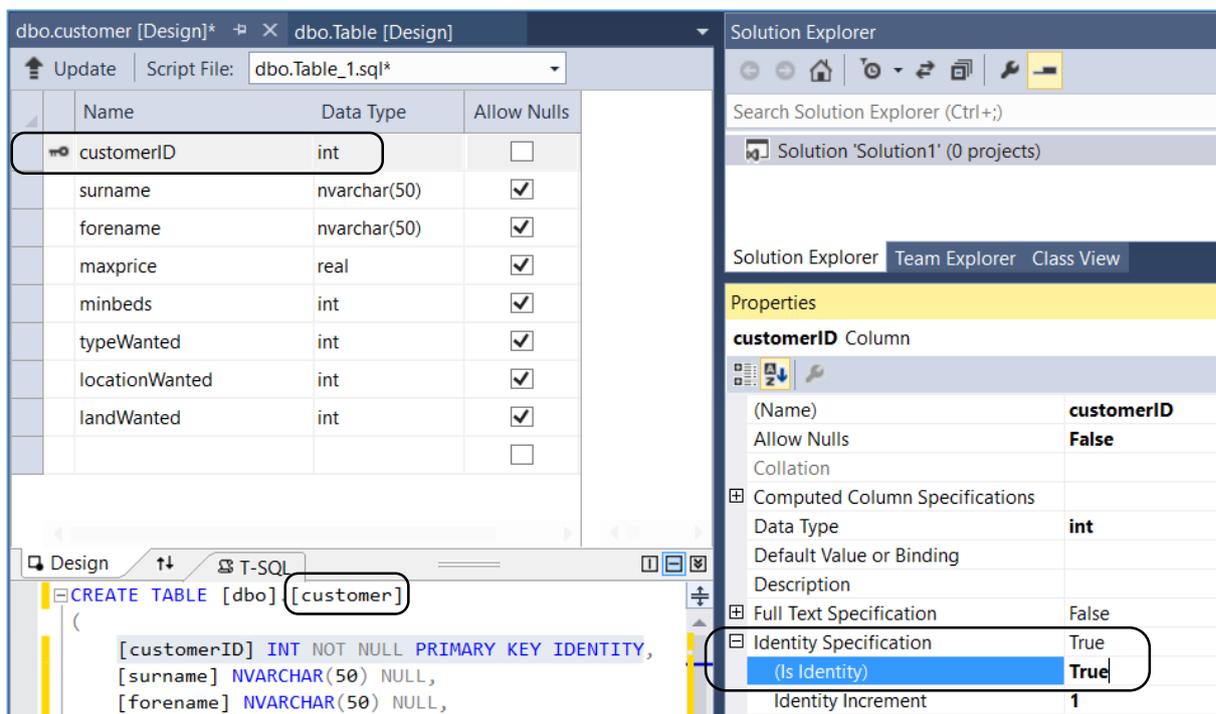
Name the table as '**house**' in the SQL window below the list of fields.

Select the '**Update**' option above the list of fields, then click the '**Update database**' button.



Close the table by clicking the small cross on the tab.

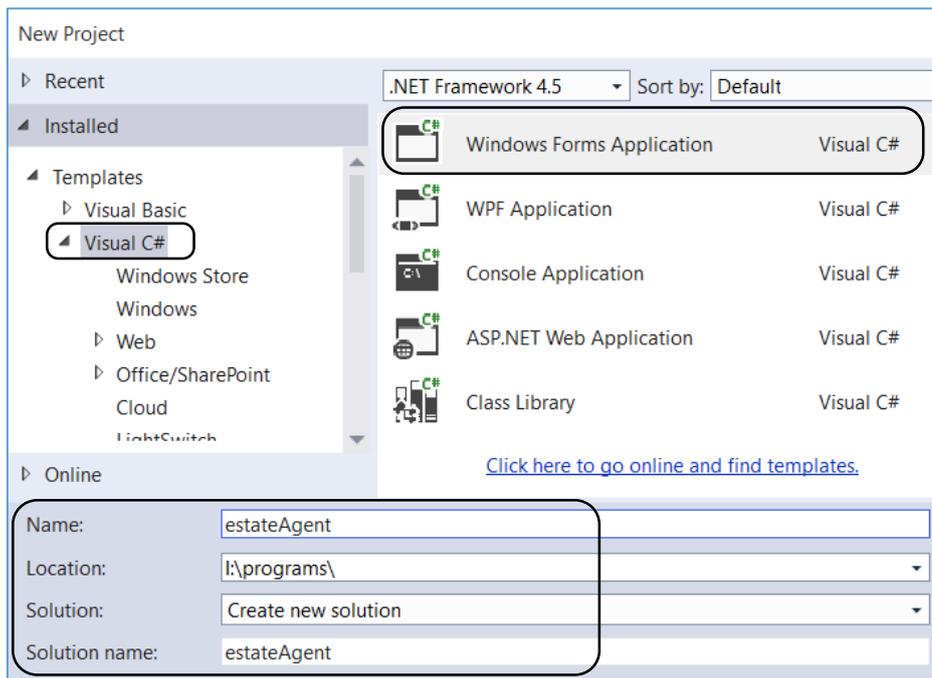
Repeat the sequence of steps above to create another table in the database. This will store details of customers and their requirements. Add the fields shown:



Set the *customerID* field to be an auto-number by setting '**Identity Specification / (Is Identity)**' to '**True**'.

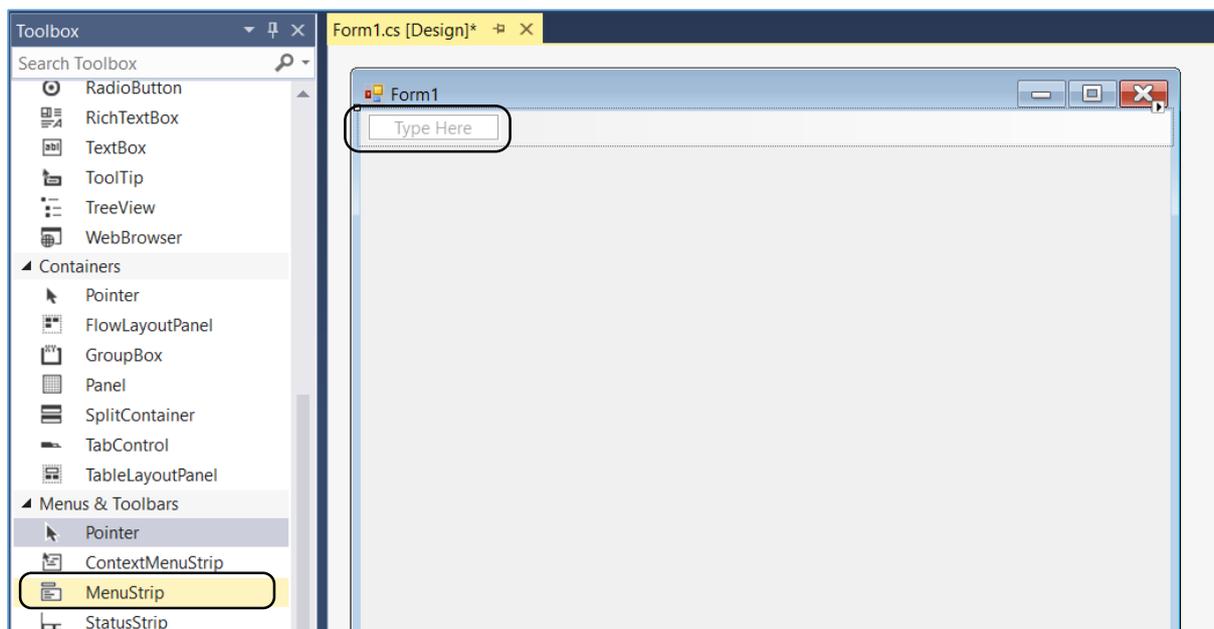
Change the table name to '**customer**' in the SQL panel. Click '**Update**' to save, then close the table by clicking the cross icon above the table window.

We can now begin the C# program which will access the database. Select '**New Project**'. Choose '**Windows Forms Application**', and set the program name to '**estateAgent**'.



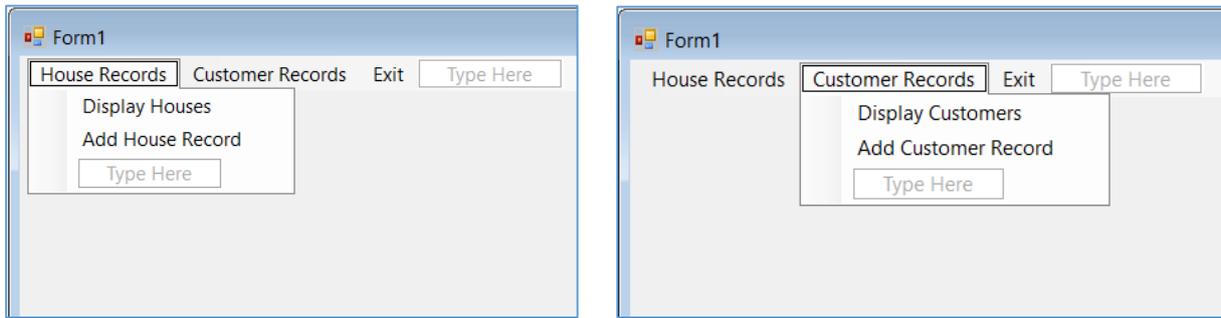
Click '**OK**' to create **Form1**.

We will set up a menu across the top of **Form1** which can be used to select the various program options. Click '**MenuStrip**' in the '**Menus & Toolbars**' section of the Toolbox, then drag the mouse to attach the menu to the top of the form.



The menu system is started by typing in the box which is displayed. As each menu item is entered, additional boxes appear alongside and below for use if required.

Build up the series of menu options shown below, by typing captions into the required boxes:

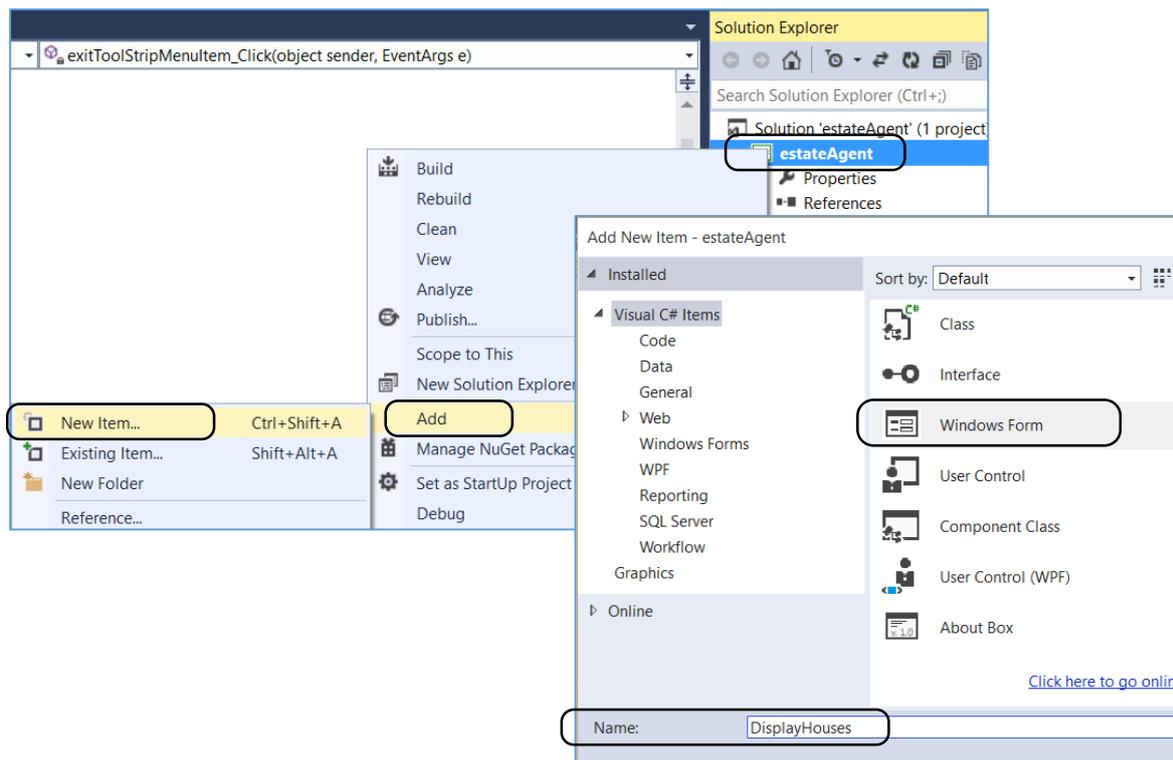


Double click the **'Exit'** menu option. An event handling method will be created. Add the line of code which will close the program.

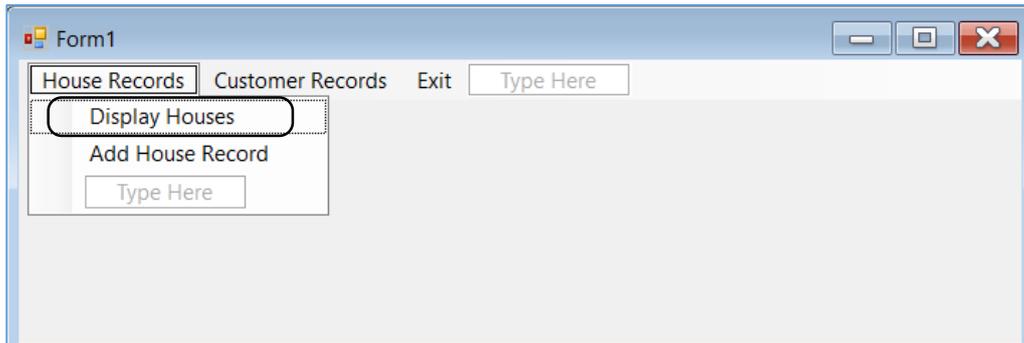
```
namespace estateAgent
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

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

Go to the **Solution Explorer** window and right click the **'estateAgent'** icon. Select **'Add / New item'**. Choose **'Windows Form'**, and set the name to **'DisplayHouses'**.



We will link this form to the menu. Double click the '*Display Houses*' option to create an event handling method:



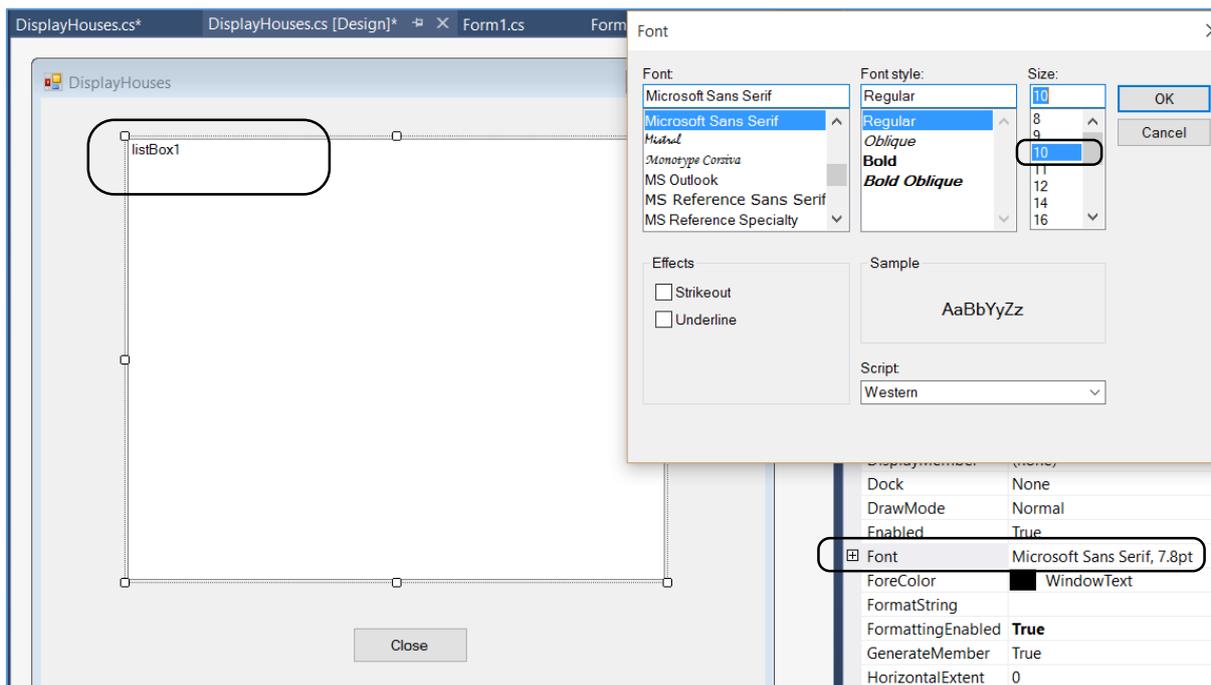
Add lines of code to open the *DisplayHouses* form:

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

private void displayHousesToolStripMenuItem_Click(object sender, EventArgs e)
{
    DisplayHouses frmDisplayHouses = new DisplayHouses();
    frmDisplayHouses.ShowDialog();
}
```

Run the program and check that the '*Display Houses*' and '*Exit*' menu options operate correctly.

We can now start to work on the *DisplayHouses* form. This will show a list of houses for sale. Add a *listBox* component to the form, and set the *font size* of the listBox to *10pt*. Also add a '*Close*' button and rename this a *btnClose*.



Double click the Close button, then a line of code to event handling method:

```
namespace estateAgent
{
    public partial class DisplayHouses : Form
    {
        public DisplayHouses()
        {
            InitializeComponent();
        }

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

We can now set up a method to load details of houses for sale. Using similar code to previous programs, we will:

- Add a '**using SqlClient**' directive.
- Specify the database location.
- Set up an empty **loadAddresses()** method. This will contain the program code to load house records from the database.
- Create a **dataSet** to receive the house data when it is loaded.
- Call the **loadAddresses()** method from **DisplayHouses()**, so that house records are loaded when the form opens.

```
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace estateAgent
{
    public partial class DisplayHouses : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf";

        public DisplayHouses()
        {
            InitializeComponent();
            loadAddresses();
        }

        DataSet dsHouses = new DataSet();

        public void loadAddresses()
        {
        }

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

We will add code to load the house records, again closely following the pattern of previous programs. Remember that the line beginning:

SqlConnection con = new SqlConnection(...

must be entered as a single line of code with no line breaks.

```
public void loadAddresses()
{
    SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
    AttachDbFilename=" + databaseLocation + "Integrated Security=True;
    Connect Timeout=30; User Instance=True");

    try
    {
        con.Open();
        SqlCommand cmHouses = new SqlCommand();
        cmHouses.Connection = con;
        cmHouses.CommandType = CommandType.Text;
        cmHouses.CommandText = "SELECT * FROM house";

        SqlDataAdapter daHouses = new SqlDataAdapter(cmHouses);
        daHouses.Fill(dsHouses);
        con.Close();
    }
    catch
    {
        MessageBox.Show("File error");
    }
}
```

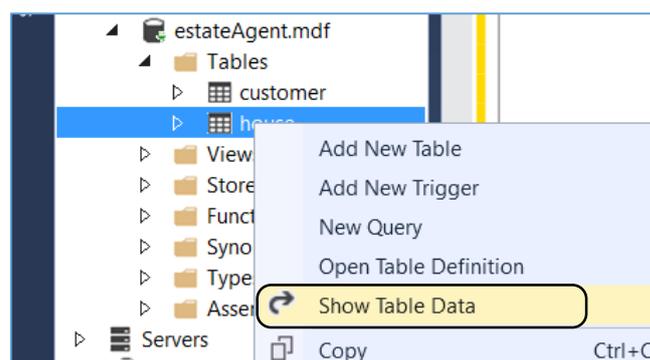
Once the house data has been transferred to the ***dataSet***, we will use a loop to access each record, picking out the address fields for display in the ***list box***:

```
SqlDataAdapter daHouses = new SqlDataAdapter(cmHouses);
daHouses.Fill(dsHouses);
con.Close();

int countRecords = dsHouses.Tables[0].Rows.Count;

for (int i = 0; i < countRecords; i++)
{
    DataRow drHouse = dsHouses.Tables[0].Rows[i];
    string houseAddress = drHouse[1] + ", " + drHouse[2];
    listBox1.Items.Add(houseAddress);
}
}
```

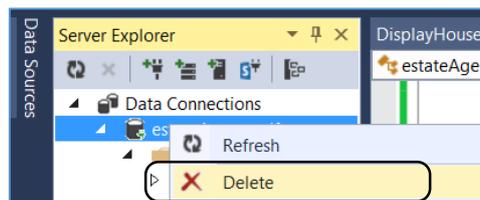
Before testing the program, it will be necessary to enter sample house data. Go to the Server Explorer, double click the '***house***' table icon, then select '***Show Table Data***'.



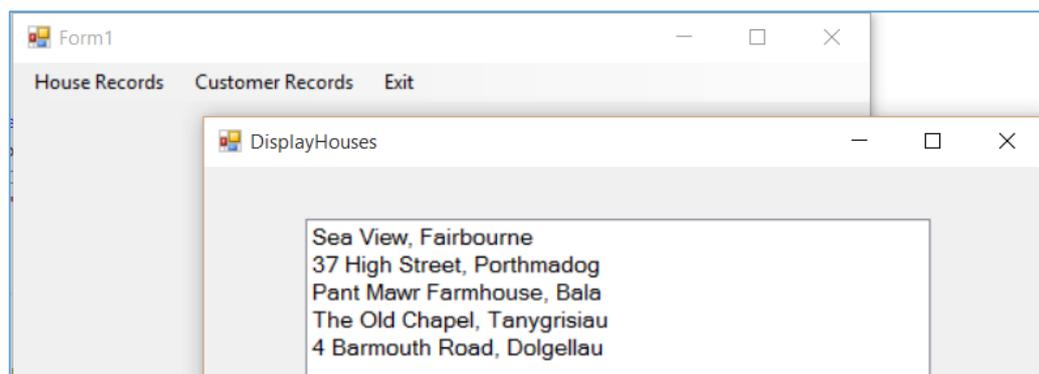
Add example test data to the table. The values for *propertyType*, *location* and *land* are code numbers which will be explained shortly.

houseID	address1	address2	price	bedrooms	propertyType	location	land
1	Sea View	Fairbourne	138000	2	3	2	1
2	37 High Street	Porthmadog	142000	3	4	1	1
3	Pant Mawr Farmhouse	Bala	464000	4	1	3	3
4	The Old Chapel	Tanygrisiau	258000	4	1	2	2
5	4 Barmouth Road	Dolgellau	380000	5	2	1	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Close the *house* table. Select '*estateAgent.mdf*' in the *Server Explorer* window, right-click and use the *Delete* option to delete the data connection. This must be done before the program is run.

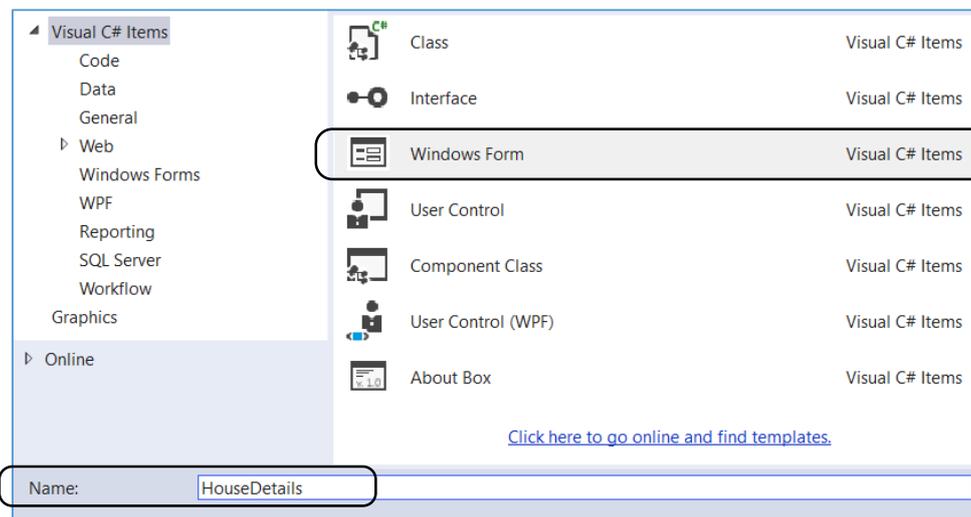


Run the program. Select the menu option to display houses. The house addresses should appear in the list box on the DisplayHouses form:



When a user clicks on one of the house addresses, we will arrange for full details of the property to be displayed.

Go to the *Solution Explorer* window, right-click the '*estateAgent*' program icon, and select '*Add / New item*'. Set up a *Windows Form* with the name '*HouseDetails*'.

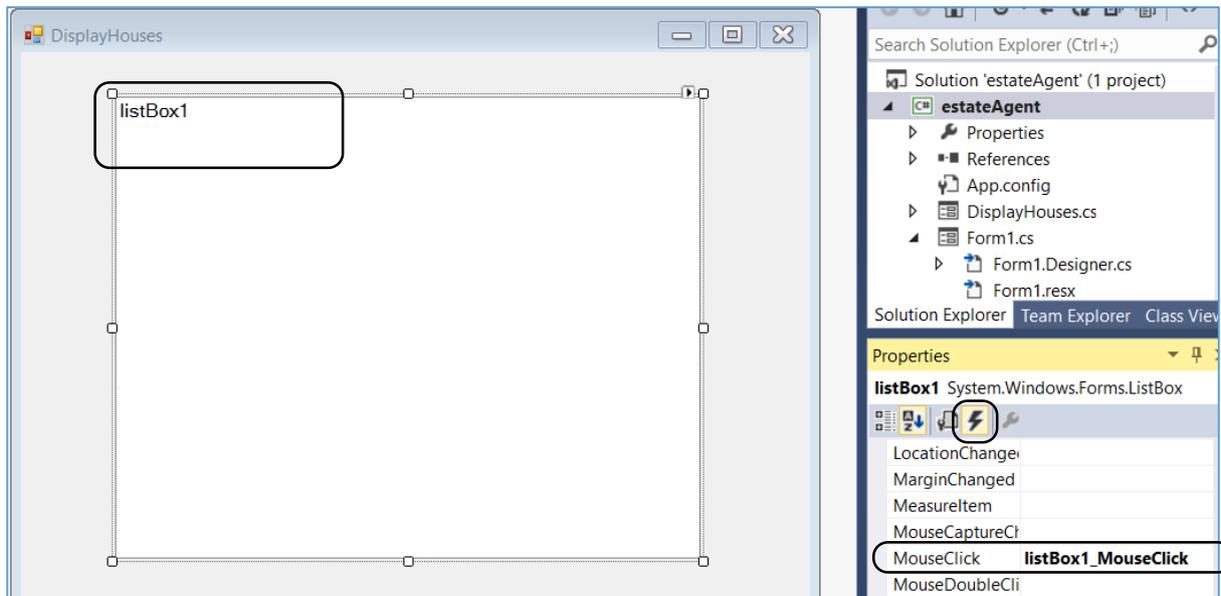


The new **HouseDetails** form will open. Right-click on the form and select '**View code**'. Add an empty method called **getHouseDetails()**. This will be used to collect and display full information about a selected house.

```
namespace estateAgent
{
    public partial class HouseDetails : Form
    {
        public HouseDetails()
        {
            InitializeComponent();
        }

        public void getHouseDetails(DataRow drHouse)
        {
        }
    }
}
```

Return to **DisplayHouses** form and select the **Design** view. Click to select the **listBox** on the form. Go to the Properties window and click the **Events** icon. Identify the **MouseClick** event in the list, then double click to create an event handler:



Add lines of code to the **listBox1_mouseClick()** method

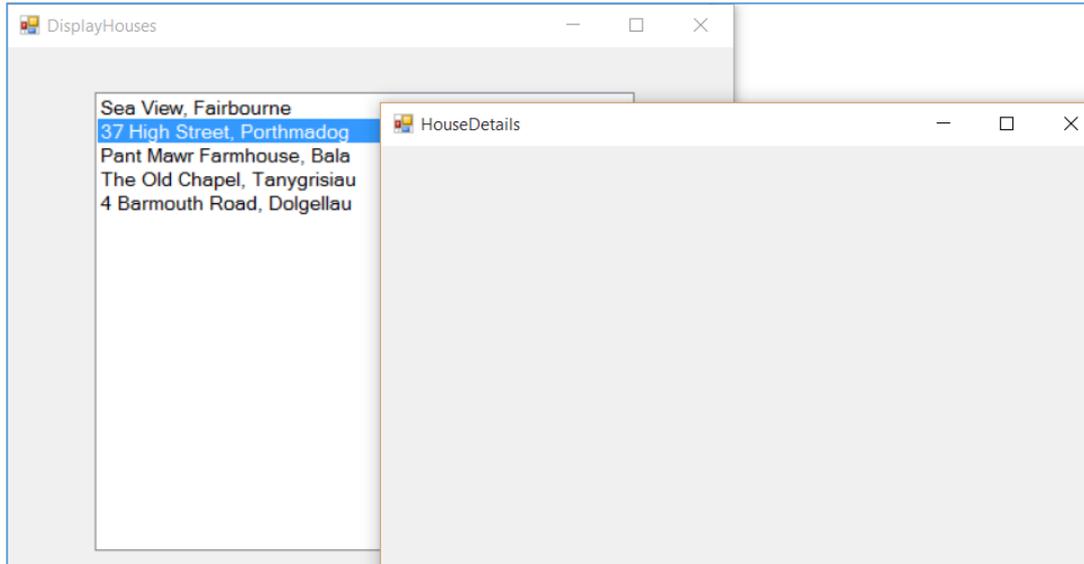
```
private void listBox1_MouseClick(object sender, MouseEventArgs e)
{
    HouseDetails frmHouseDetails = new HouseDetails();
    int houseSelected = listBox1.SelectedIndex;
    DataRow drHouseWanted = dsHouses.Tables[0].Rows[houseSelected];

    frmHouseDetails.getHouseDetails(drHouseWanted);
    frmHouseDetails.ShowDialog();
    this.Close();
}
```

When a house address is clicked in the listBox, this method will carry out a series of tasks:

- It will find the position in the list of the selected house.
- It will collect the corresponding **house record** from the **dataSet**.
- It will then transfer the house record to the **HouseDetails form** where it can be displayed.

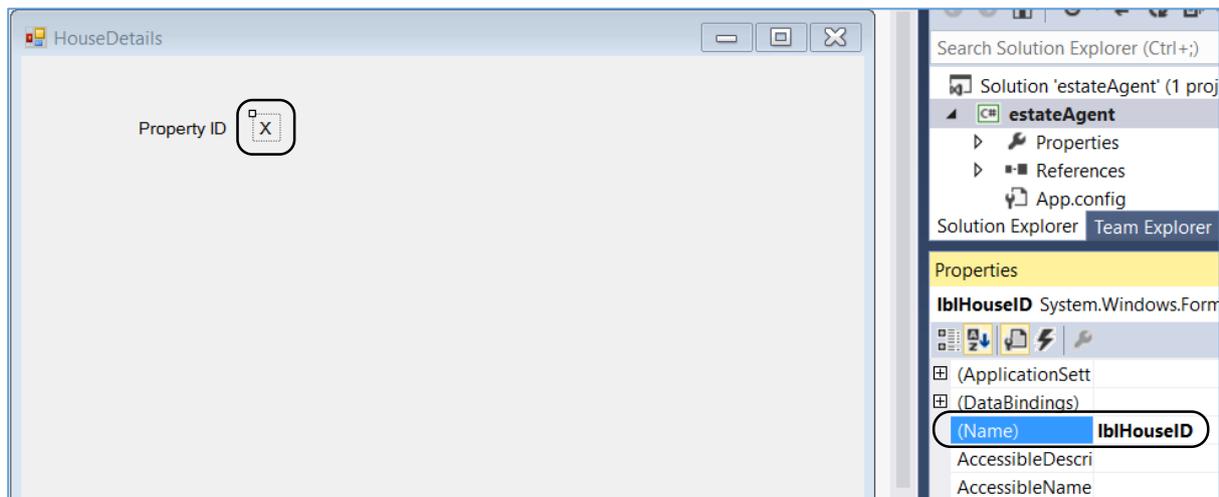
Run the program and check that the **HouseDetails** window opens correctly when a house address is clicked in the listBox. The actual details of the house are not yet displayed.



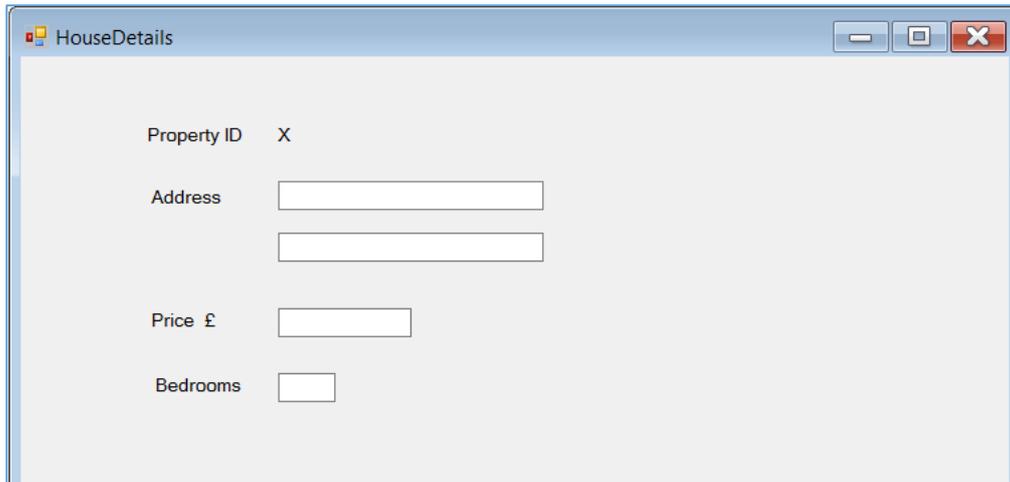
Close the program and open the **HouseDetails form**, where we will display the house data.

The first item to show is the **HouseID**. Unlike other details of the house, this field should not be editable. We will therefore display it with a **Label** component. Give this the name '**lblHouseID**'.

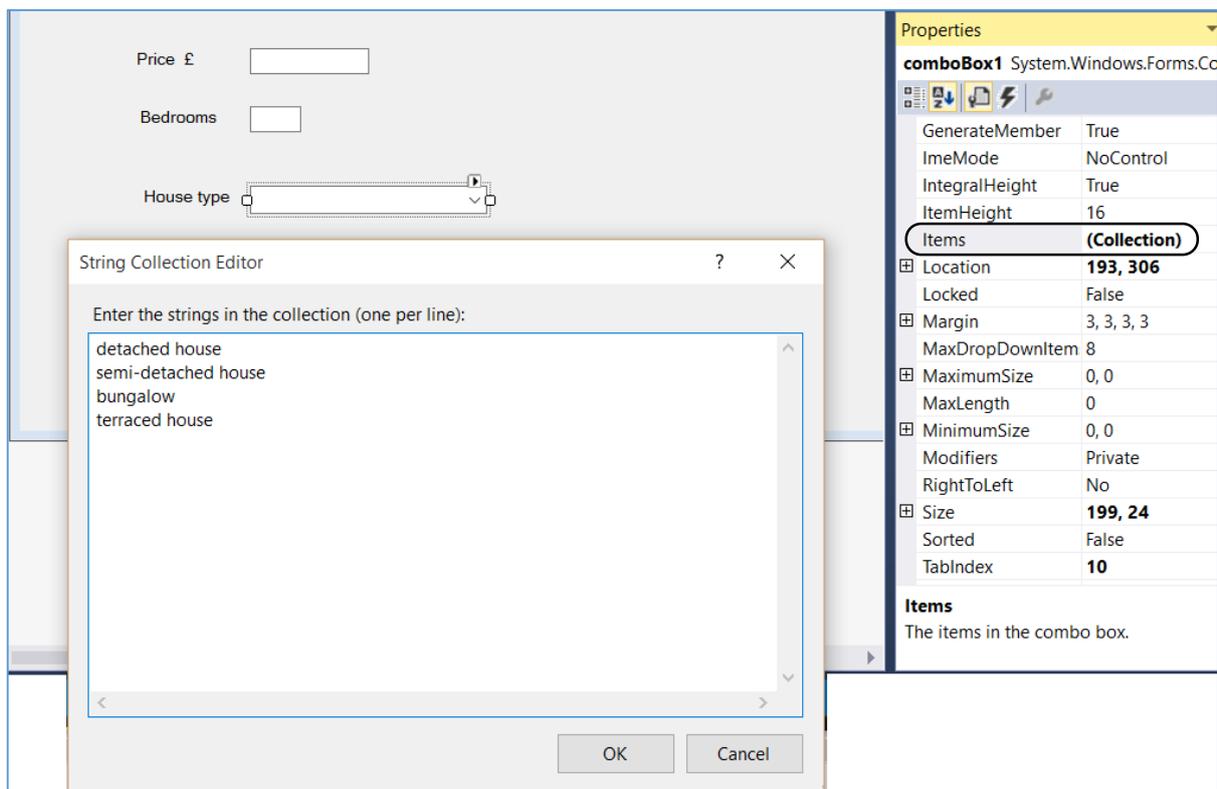
Add the text '**X**' for the label at this stage. This will be replaced by the actual houseID when the program runs.



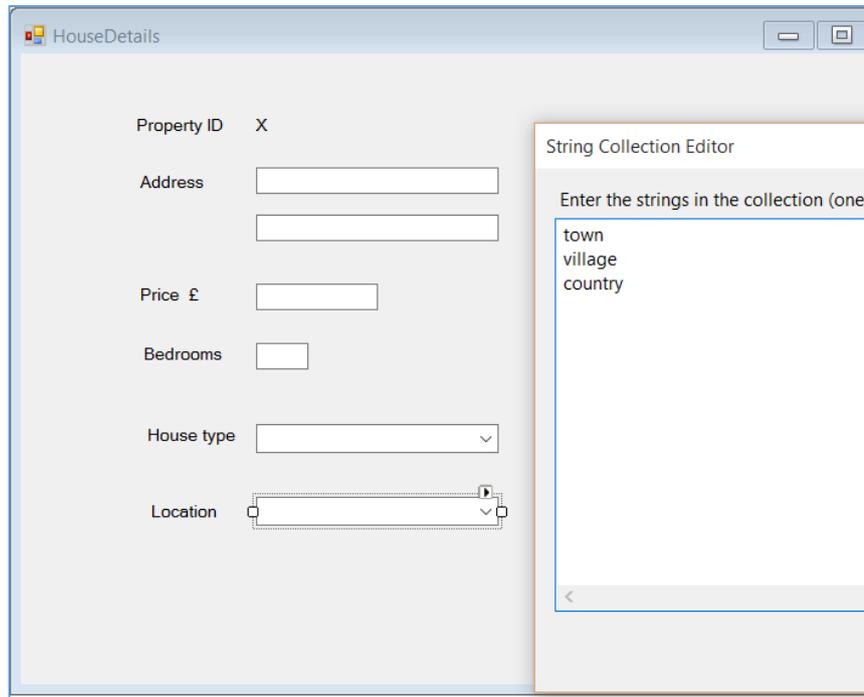
Continue to build the display screen by adding labels and textboxes. The textboxes should be renamed as: *txtAddress1*, *txtAddress2*, *txtPrice*, *txtBedrooms*:



Insert a **ComboBox** component for the *houseType* field. We will provide a drop down list of house types. To do this, select the comboBox and go to the Properties window. Find 'Items' and click to open a String Editor window. Enter the list of house types as shown, then click the OK button.

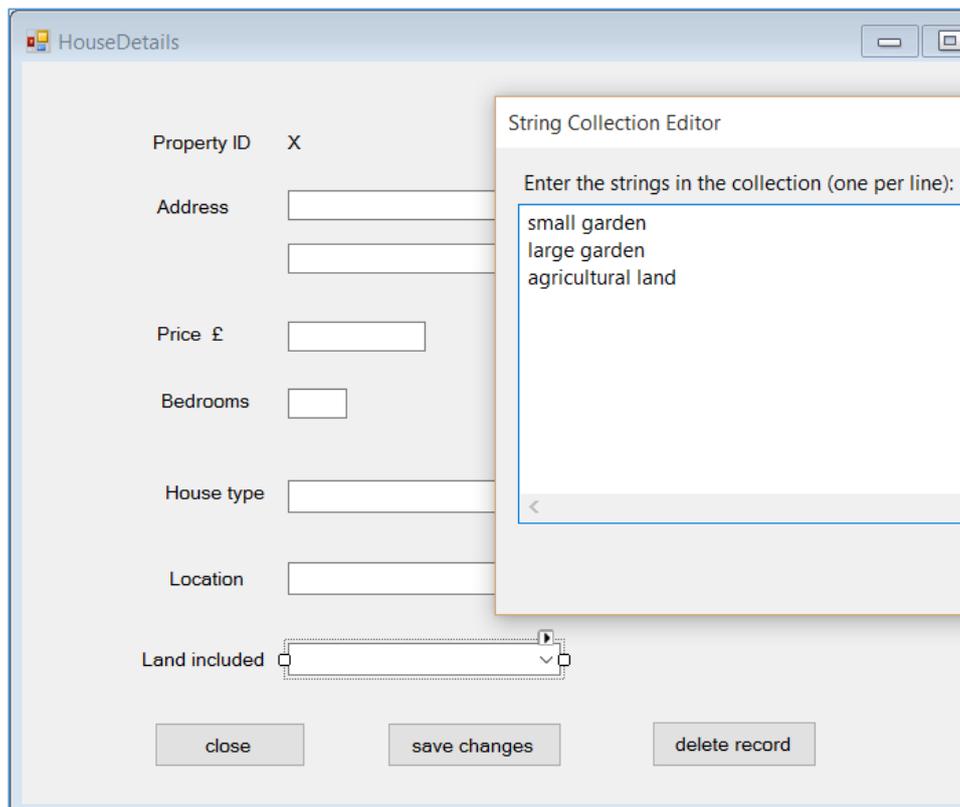


Add a comboBox for **Location**, and enter the list of location options:



Add a comboBox for the '**Land included**' field and enter the options for the drop down list

Complete the form by adding three buttons for the options '**close**', '**save changes**' and '**delete record**'. Name the buttons as **btnClose**, **btnUpdate** and **btnDelete**.



Return to the program listing for the **HouseDetails** form and add lines of code to the **getHouseDetails()** method. This method receives the selected house record from the previous form as the parameter **drHouse**. The individual fields are extracted from the record, then displayed in the textBoxes or comboBoxes:

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

    public void getHouseDetails(DataRow drHouse)
    {
        lblHouseID.Text = Convert.ToString(drHouse[0]);
        txtAddress1.Text = Convert.ToString(drHouse[1]);
        txtAddress2.Text = Convert.ToString(drHouse[2]);
        txtPrice.Text = String.Format("{0:0,0}", drHouse[3]);
        txtBedrooms.Text = Convert.ToString(drHouse[4]);
        comboBox1.SelectedIndex = Convert.ToInt16(drHouse[5]) - 1;
        comboBox2.SelectedIndex = Convert.ToInt16(drHouse[6]) - 1;
        comboBox3.SelectedIndex = Convert.ToInt16(drHouse[7]) - 1;
    }
}
```

Run the program. Go to the **'Display Houses'** menu option, then select a house. The House Details form should open to display the full set of fields for the house record:

The screenshot shows a Windows application window titled 'DisplayHouses' with a sub-window titled 'HouseDetails'. The 'HouseDetails' form displays the following information:

- Property ID: 1
- Address: Sea View (txtAddress1) and Fairbourne (txtAddress2)
- Price £: 138,000 (txtPrice)
- Bedrooms: 2 (txtBedrooms)
- House type: bungalow (comboBox1)
- Location: village (comboBox2)
- Land included: small garden (comboBox3)

At the bottom of the form, there are three buttons: 'close', 'save changes', and 'delete record'. In the background, a list of houses is visible, with 'Sea View, Fair' selected.

'House Type', **'Location'** and **'Land included'** are stored in the house record as code numbers, e.g. detached house = 1, semi-detached house=2, ... These values are used to select the appropriate list item for display in each of the comboBoxes.

Close the program windows and return to the program listing for the *HouseDetails* page. We will now turn our attention to updating the house record if the user wishes to change any of the details.

Add a *'using SqlClient'* directive at the start of the program, and specify the location of the database:

```
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace estateAgent
{
    public partial class HouseDetails : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf";

        public HouseDetails()
        {
            InitializeComponent();
        }
    }
}
```

Double click the *'save changes'* button to create a *btnUpdate_Click()* method.

Add lines of program code which will take the values from the textBoxes and comboBoxes and store them temporarily as variables of the correct data type, ready for updating the house record. We will also add the lines of code for connection to the database:

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    int HouseID = Convert.ToInt16(lblHouseID.Text);
    string Address1 = txtAddress1.Text;
    string Address2 = txtAddress2.Text;
    double Price = Convert.ToDouble(txtPrice.Text);
    int Bedrooms = Convert.ToInt16(txtBedrooms.Text);
    int Housetype = comboBox1.SelectedIndex + 1;
    int Location = comboBox2.SelectedIndex + 1;
    int Land = comboBox3.SelectedIndex + 1;

    SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
    AttachDbFilename=" + databaseLocation + "Integrated Security=True;
    Connect Timeout=30; User Instance=True");
}
```

The addition of 1 to the comboBox index values comes about because items in a comboBox are numbered from zero, whereas it is more sensible for our code numbers to begin at 1. For example, in the 'Location' field:

town = code 1

town = listBox item 0

village = code 2

village = listBox item 1

etc...

We can now add the code to open the database and update the house record. This is similar to the code which we have written previously to load data, but this time makes use of an **UPDATE** command in SQL.

```

SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
AttachDbFilename=" + databaseLocation + "Integrated Security=True;
Connect Timeout=30; User Instance=True");

try
{
    con.Open();
    SqlCommand cmHouses = new SqlCommand();
    cmHouses.Connection = con;
    cmHouses.CommandType = CommandType.Text;
    cmHouses.CommandText = "UPDATE house SET address1='" + Address1
        + "', address2='" + Address2 + "', price='" + Price + "', bedrooms='"
        + Bedrooms + "', propertyType='" + Housetype + "', location='"
        + Location + "', land='" + Land + "' WHERE houseID='" + HouseID + "'";
    cmHouses.ExecuteNonQuery();
    con.Close();
    this.Close();
}
catch
{
    MessageBox.Show("File error");
}
}

```

Run the program. Select a house and make some changes to the details, such as the address, price or land included. Click **'save changes'**.

Return to the list of properties for sale. Click the house address to reopen the **HouseDetails** form. Check that your changes were made correctly. Close the program windows and return to the program listing for the **HouseDetails** page.

If all is well, we can now program the **'delete record'** option. It is important to give the user an option to cancel if they have clicked the **'delete record'** button by accident. We will do this by making a message box appear before the record is actually deleted from the database.

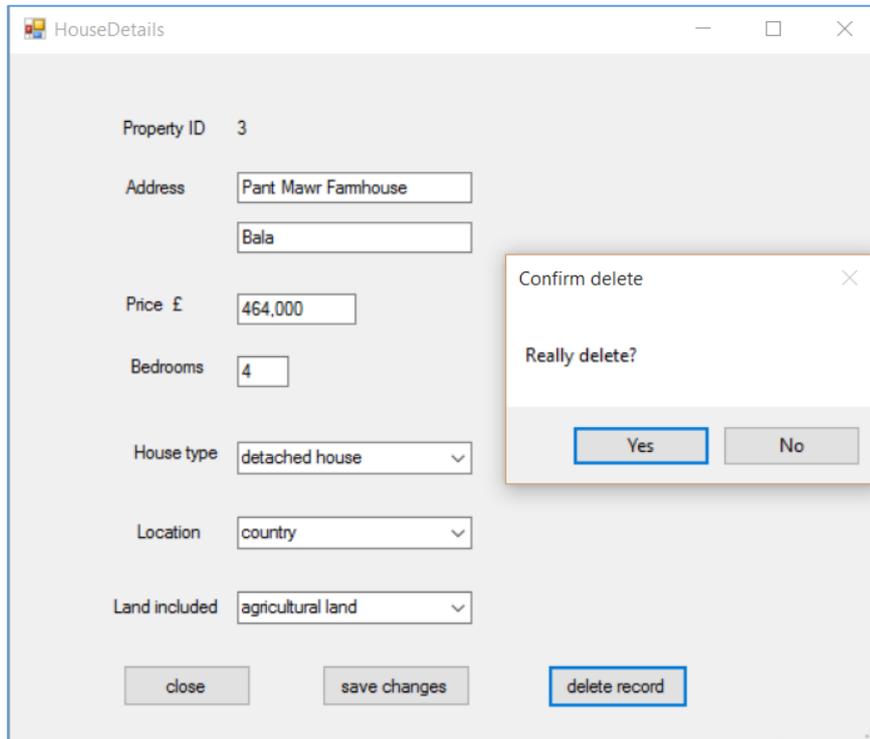
Double click the **'delete record'** button to create an event method, then add code to open a message box:

```

private void btnDelete_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Really delete?", "Confirm delete",
        MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
    }
}

```

Run the program. Select a house, then click the **'delete record'** button. Check that the confirm message appears. Close the program and return to the program listing for the **HouseDetails** page.



We will now add the code which will actually delete the record from the database if the user answers **'Yes'**. This is again very similar to previous database code we have written, but this time uses the **DELETE** command in SQL. The correct record for deletion is selected by means of the **houseID** value.

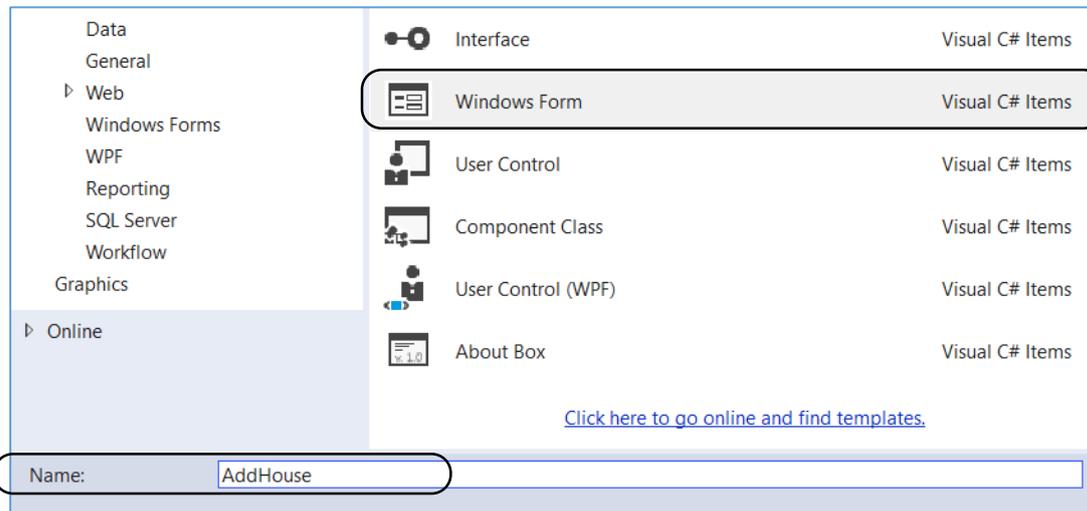
```
private void btnDelete_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Really delete?", "Confirm delete",
        MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        int HouseID = Convert.ToInt16(lblHouseID.Text);

        SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
        AttachDbFilename=" + databaseLocation + "Integrated Security=True;
        Connect Timeout=30; User Instance=True");

        try
        {
            con.Open();
            SqlCommand cmHouses = new SqlCommand();
            cmHouses.Connection = con;
            cmHouses.CommandType = CommandType.Text;
            cmHouses.CommandText = "DELETE house WHERE houseID='" + HouseID + "'";
            cmHouses.ExecuteNonQuery();
            con.Close();
            this.Close();
        }
        catch
        {
            MessageBox.Show("File error");
        }
    }
}
```

The final operation we need to carry out on the *'house'* table is to **add new records**. We will postpone testing the *'delete record'* option until this *'add record'* function has been completed.

We will require another **Windows Form** where house details can be entered. Go to the **Solution Explorer** window and right-click on the *estateAgent* program icon. Select **'Add / New item'**. Choose **'Windows Form'**, and give the name **'AddHouse'**:



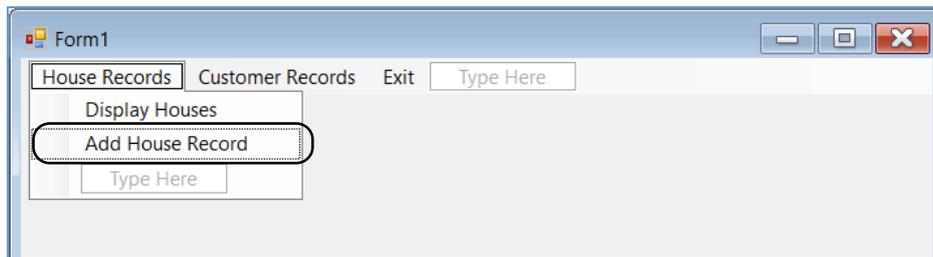
Open the **AddHouse** form and add labels, textboxes and comboBoxes to create a data entry form. You may find it easiest to copy and paste the required group of components from the *HouseDetails* form which you set up earlier. **Buttons** will be required to **cancel** or **save** the house record:

The image shows a Windows Form titled 'AddHouse'. It has a light gray background and a standard Windows window border with minimize, maximize, and close buttons. The form contains the following controls:

- Address:** Two stacked text boxes.
- Price £:** A single text box.
- Bedrooms:** A single text box.
- House type:** A dropdown menu.
- Location:** A dropdown menu.
- Land included:** A dropdown menu.
- Buttons:** Two buttons at the bottom: 'cancel' and 'save house record'.

Name the buttons as **btnCancel** and **btnSave**.

Return to **Form1** and link the **AddHouse** form to the menu system by double clicking the '**Add House Record**' menu option:



Add lines of code to open the AddHouse form:

```
private void displayHousesToolStripMenuItem_Click(object sender, EventArgs e)
{
    DisplayHouses frmDisplayHouses = new DisplayHouses();
    frmDisplayHouses.ShowDialog();
}

private void addHouseRecordToolStripMenuItem_Click(object sender, EventArgs e)
{
    AddHouse frmAddHouse = new AddHouse();
    frmAddHouse.ShowDialog();
}
```

Return to the '**AddHouse**' form. Double click the '**cancel**' button to create an event procedure and add the **Close()** command.

Go to the start of the program listing and insert the '**using SqlClient**' directive and the database location.

```
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace estateAgent
{
    public partial class AddHouse : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf;";

        public AddHouse()
        {
            InitializeComponent();
        }

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

Double click the '*save house record*' button to produce an event method.

Add code which will collect the required data from the textboxes and comboBoxes, ready for transfer to a database record. We will also set up the connection to the database.

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

private void btnSave_Click(object sender, EventArgs e)
{
    string Address1 = txtAddress1.Text;
    string Address2 = txtAddress2.Text;
    double Price = Convert.ToDouble(txtPrice.Text);
    int Bedrooms = Convert.ToInt16(txtBedrooms.Text);
    int Housetype = comboBox1.SelectedIndex + 1;
    int Location = comboBox2.SelectedIndex + 1;
    int Land = comboBox3.SelectedIndex + 1;
    SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
    AttachDbFilename=" + databaseLocation + "Integrated Security=True;
    Connect Timeout=30; User Instance=True");
}
```

We then add the code to save the new record into the database. This uses the **INSERT** command in SQL. Notice that no value is included for the **houseID** field. We specified this as an auto-number field, so the value will be allocated automatically by the database.

```
SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
AttachDbFilename=" + databaseLocation + "Integrated Security=True;
Connect Timeout=30; User Instance=True");

try
{
    con.Open();
    SqlCommand cmHouses = new SqlCommand();
    cmHouses.Connection = con;
    cmHouses.CommandType = CommandType.Text;
    cmHouses.CommandText = "INSERT INTO house(address1, address2, price,
        bedrooms, propertyType, location, land)
        VALUES ('" + Address1 + "', '" + Address2 + "', '" + Price + "', '"
        + Bedrooms + "', '" + Housetype + "', '" + Location + "', '"
        + Land + "')";
    cmHouses.ExecuteNonQuery();
    con.Close();
    Close();
}
catch
{
    MessageBox.Show("File error");
}
```

Please note that the lines beginning

```
SqlConnection con = new SqlConnection(...  
cmHouses.CommandText = "INSERT INTO...
```

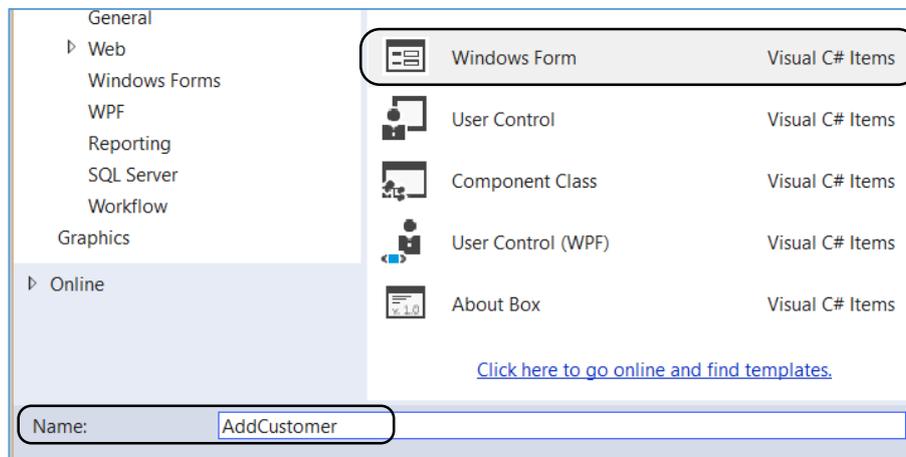
should each be entered as a single line of code with no line breaks.

Run the program and check that a new house record can be added correctly to the **'house'** table. Click the house list to display the record details. Check that the new record can then be deleted by clicking the **'delete record'** button.

We have now completed the **houses** section of the database program, and can turn our attention to the **customers** of the Estate Agent.

When registering with the company, potential buyers will be asked to specify the **maximum price** they are willing to pay for a property, the **minimum number of bedrooms** which they require, and any preferences concerning the **type of property, location or land included**. This information can be used by the Estate Agent to select suitable properties which might be of interest to the customer.

Add a **Windows Form** and give this the name **'AddCustomer'**. It will be used for entering details to create new customer records.



Add textBoxes as show below, naming these as **txtSurname**, **txtForename**, **txtMaxprice** and **txtMinbeds**.

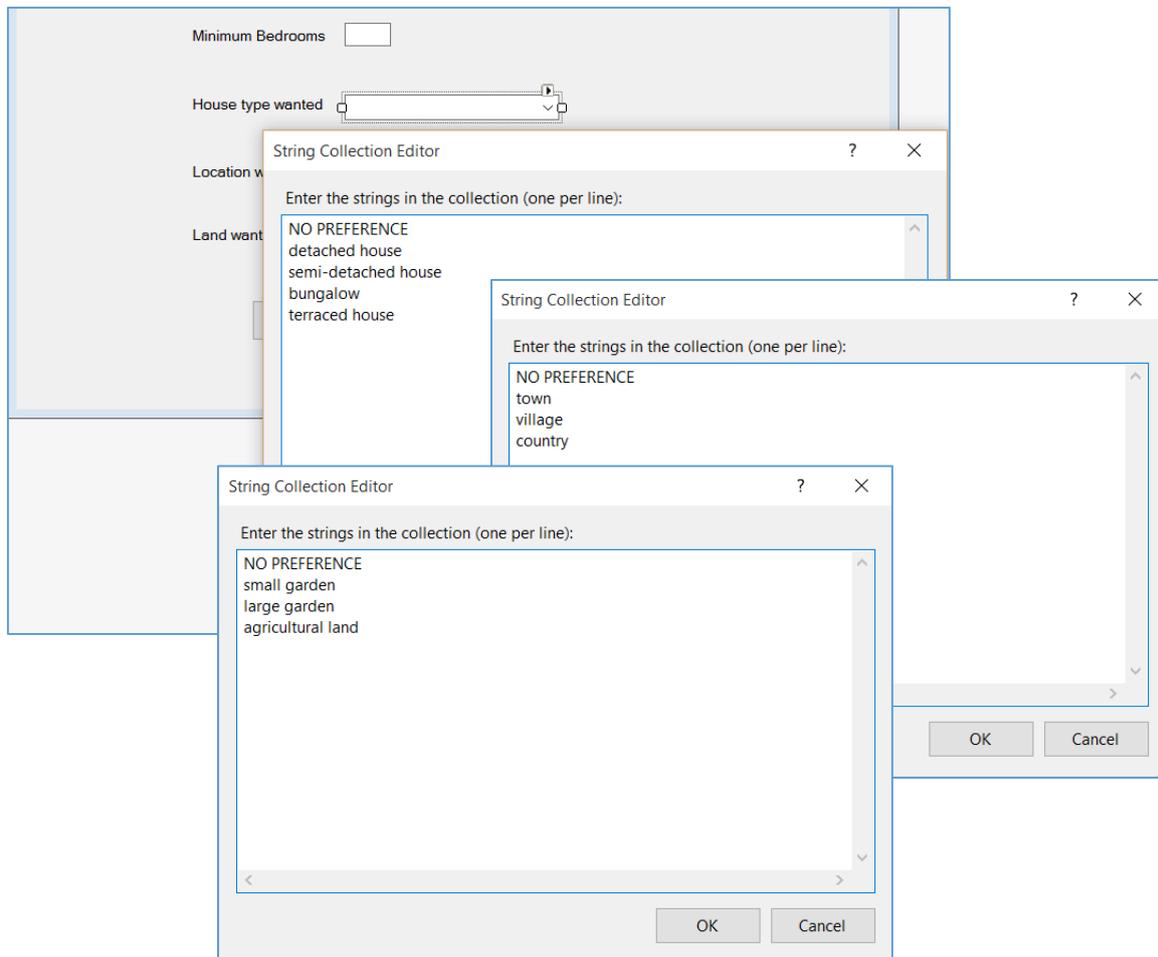
Add comboBoxes for **'House type wanted'**, **'Location wanted'**, and **'Land wanted'**. Buttons will be needed for **'cancel'** and **'save customer record'** options. Name these as **btnCancel** and **btnSave**.

The screenshot shows the 'AddCustomer' form with the following fields and buttons:

- Surname:
- Forename:
- Maximum Price £:
- Minimum Bedrooms:
- House type wanted:
- Location wanted:
- Land wanted:
- cancel:
- save customer record:

We need to enter options for the comboBox drop down lists. These will list the property types, locations and land descriptions in a similar way to the house records, but we will also include a NO PREFERENCE option at the start of each list. If the customer specifies 'no preference', for example in house type, then this field will be ignored when searching for suitable properties.

Select each of the comboBoxes in turn, then go to the **Properties** window and click to the right of **Items** to open the **String Collection Editor**. Enter the option lists as shown below



Return to **Form1**. Link the **'AddCustomer'** form to the menu system by double clicking the **'Add Customer Record'** menu option and adding lines of code to the event method:

```
private void addHouseRecordToolStripMenuItem_Click(object sender, EventArgs e)
{
    AddHouse frmAddHouse = new AddHouse();
    frmAddHouse.ShowDialog();
}

private void addCustomerRecordToolStripMenuItem_Click(object sender, EventArgs e)
{
    AddCustomer frmAddCustomer = new AddCustomer();
    frmAddCustomer.ShowDialog();
}
```

Double click the **'save customer record'** button to create an event method. Add code to transfer data from the textBoxes and comboBoxes into variables, ready for saving to the database. Go to the top of the program listing, and add the **'using SqlClient'** directive and the database location.

```
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace estateAgent
{
    public partial class AddCustomer : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf";

        public AddCustomer()
        {
            InitializeComponent();
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            string Surname = txtSurname.Text;
            string Forename = txtForename.Text;
            double Maxprice = Convert.ToDouble(txtMaxprice.Text);
            int Minbedrooms = Convert.ToInt16(txtMinbeds.Text);
            int Typewanted = comboBox1.SelectedIndex;
            int Locationwanted = comboBox2.SelectedIndex;
            int Landwanted = comboBox3.SelectedIndex;

            SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
            AttachDbFilename=" + databaseLocation + "Integrated Security=True;
            Connect Timeout=30; User Instance=True");
        }
    }
}
```

The final step is to save the record to the database. Add code to carry out the **INSERT** command in SQL. Notice again that the **auto-number** field **CustomerID** is not included in the list of values:

```
SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
AttachDbFilename=" + databaseLocation + "Integrated Security=True;
Connect Timeout=30; User Instance=True");

try
{
    con.Open();
    SqlCommand cmCustomers = new SqlCommand();
    cmCustomers.Connection = con;
    cmCustomers.CommandType = CommandType.Text;
    cmCustomers.CommandText =
        "INSERT INTO customer(surname, forename, maxprice, minbeds, typewanted,
        locationWanted, landWanted) VALUES ('" + Surname + "','" + Forename
        + "','" + Maxprice + "','" + Minbedrooms + "','" + Typewanted
        + "','" + Locationwanted + "','" + Landwanted + "')";
    cmCustomers.ExecuteNonQuery();
    con.Close();
    this.Close();
}
catch
{
    MessageBox.Show("File error");
}
}
```

Run the program. Enter test data for customers:

Surname	Forename	Max price	Min beds	House type	Location	Land
Jenkins	Aled	180,000	3	No preference	No preference	No preference
Humphries	Stuart	400,000	2	No preference	Village	Large garden
Andrews	Ian	600,000	3	No preference	No preference	Agricultural
Edwards	Elisabeth	500,000	2	Detached	No preference	No preference
Pritchard	Tom	550,000	2	Bungalow	Village	No preference

Go to the Server Explorer and check that the records have been inserted into the '**customer**' table. Notice that the preferences for **house type**, **location** and **land** will be shown as code numbers, with zero representing 'NO PREFERENCE'. The customerID values have been allocated automatically. Correct any errors in the table, then right-click on the '**Data Connections**' icon and delete the connection.

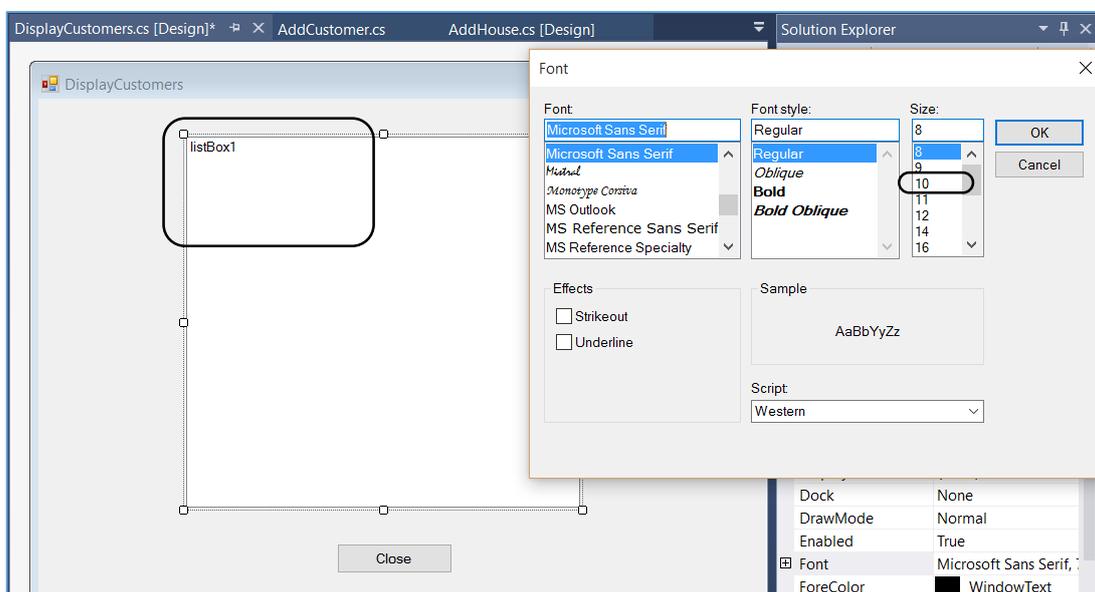
customerID	surname	forename	maxprice	minbeds	typeWanted	locationWanted	landWanted
1	Jenkins	Aled	180000	3	0	0	0
2	Humphries	Stuart	400000	2	0	2	2
4	Andrews	Ian	600000	3	0	0	3
5	Edwards	Elisabeth	500000	2	1	0	0
6	Pritchard	Tom	550000	2	3	2	0

We will now produce a customer display option. Add a new **Windows Form** and give this the name '**DisplayCustomers**'. Go to **Form1** and link the new form to the '**Display Customers**' menu option:

```
private void addCustomerRecordToolStripMenuItem_Click(object sender, EventArgs e)
{
    AddCustomer frmAddCustomer = new AddCustomer();
    frmAddCustomer.ShowDialog();
}

private void displayCustomersToolStripMenuItem_Click(object sender, EventArgs e)
{
    DisplayCustomers frmDisplayCustomers = new DisplayCustomers();
    frmDisplayCustomers.ShowDialog();
}
```

Add a list box to the **DisplayCustomers** form, and set the **font size to 10 point**. Add a '**Close**' button.



We will add code to the *DisplayCustomers* form to include the *'using SqlClient'* directive and specify the database location. Create an empty *loadCustomerNames()* method, and call this from the *DisplayCustomers()* method. Produce a *dataSet* to hold the customer records when they are loaded.

```

using System.Text;
using System.Windows.Forms;

using System.Data.SqlClient;

namespace estateAgent
{
    public partial class DisplayCustomers : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf";

        public DisplayCustomers()
        {
            InitializeComponent();
            loadCustomerNames();
        }

        DataSet dsCustomers = new DataSet();

        public void loadCustomerNames()
        {
        }
    }
}

```

The code to load customer records from the database can now be added.

```

public void loadCustomerNames()
{
    SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
    AttachDbFilename=" + databaseLocation + "Integrated Security=True;
    Connect Timeout=30; User Instance=True");
    try
    {
        con.Open();
        SqlCommand cmCustomers = new SqlCommand();
        cmCustomers.Connection = con;
        cmCustomers.CommandType = CommandType.Text;
        cmCustomers.CommandText = "SELECT * FROM customer";
        SqlDataAdapter daCustomers = new SqlDataAdapter(cmCustomers);
        daCustomers.Fill(dsCustomers);
        con.Close();

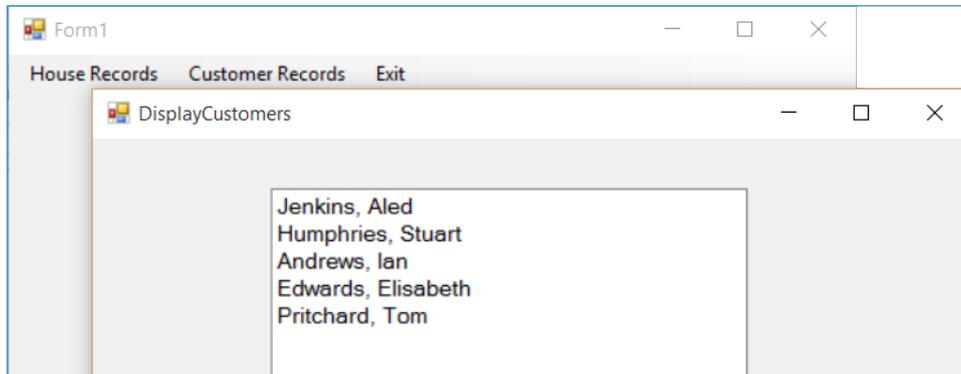
        int countRecords = dsCustomers.Tables[0].Rows.Count;

        for (int i = 0; i < countRecords; i++)
        {
            DataRow drCustomer = dsCustomers.Tables[0].Rows[i];
            string customerName = drCustomer[1] + ", " + drCustomer[2];
            listBox1.Items.Add(customerName);
        }
    }
    catch
    {
        MessageBox.Show("File error");
    }
}

```

Notice how the loop takes each **data row** in turn from the whole **data set**, then extracts the **surname** and **forename** fields. These are assembled together, separated by a comma, for display in the list box.

Run the program and check that the customer names are displayed correctly.



As with the houses earlier, we will create another form to display the full customer record when a name is selected from the list box.

Add a new **Windows Form** and name this '**CustomerDetails**'. Add labels, textBoxes and comboBoxes to the form as shown below. To save time, copy and paste the required components from the **AddCustomer** form which you created earlier, keeping the component names unaltered.

Include a label for display of the **customerID**. Give this the name '**lblCustomerID**', and set the text initially to '**X**'. Add buttons to **close** the form, **save changes** to the record, **delete** the record, and to **search for suitable properties** for this customer, naming these as **btnClose**, **btnSave**, **btnDelete** and **btnSearch**.

Go to the '**CustomerDetails**' program page and add lines of code to include the '**using SqlConnection**' directive, and to specify the database location. Create a **getCustomerDetails()** method which will accept a customer record, set up variables from each of the fields, then display the data using the screen components.

```

using System.Text;
using System.Windows.Forms;

using System.Data.SqlClient;

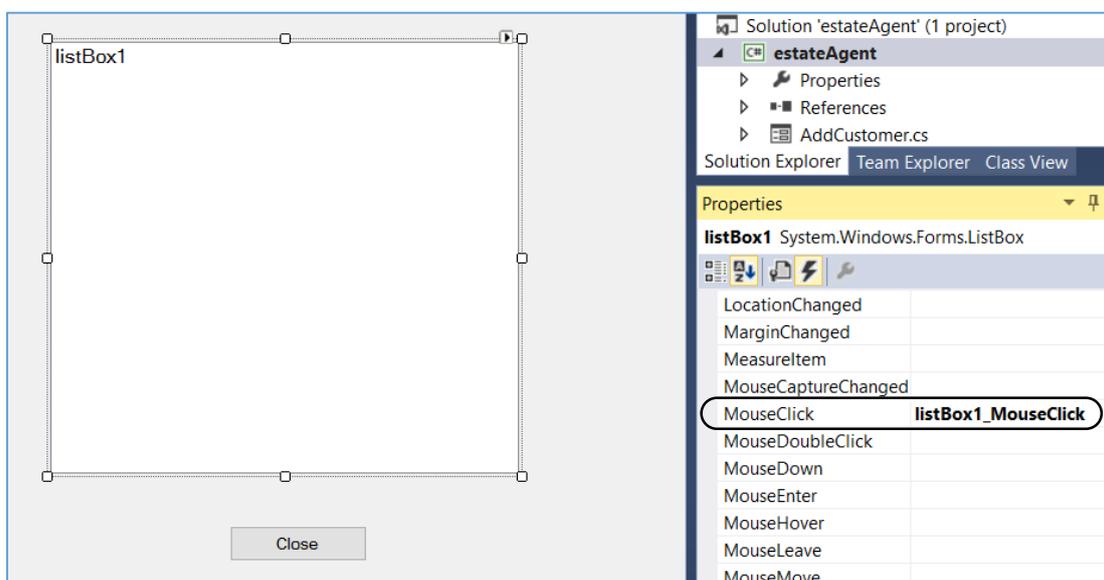
namespace estateAgent
{
    public partial class CustomerDetails : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf;";

        public CustomerDetails()
        {
            InitializeComponent();
        }

        public void getCustomerDetails(DataRow drCustomer)
        {
            lblCustomerID.Text = Convert.ToString(drCustomer[0]);
            txtSurname.Text = Convert.ToString(drCustomer[1]);
            txtForename.Text = Convert.ToString(drCustomer[2]);
            txtMaxprice.Text = String.Format("{0:0,0}", drCustomer[3]);
            txtMinbeds.Text = Convert.ToString(drCustomer[4]);
            comboBox1.SelectedIndex = Convert.ToInt16(drCustomer[5]);
            comboBox2.SelectedIndex = Convert.ToInt16(drCustomer[6]);
            comboBox3.SelectedIndex = Convert.ToInt16(drCustomer[7]);
        }
    }
}

```

Return to the **DisplayCustomers** form and select the **listBox**. Go to the Properties window and click the **Events** icon. Locate the **MouseClick** event, and double click to create an event method.



Add code to the *mouseClick()* method:

```
private void listBox1_MouseClick(object sender, MouseEventArgs e)
{
    CustomerDetails frmCustomerDetails = new CustomerDetails();
    int customerSelected = listBox1.SelectedIndex;

    DataRow drCustomerWanted = dsCustomers.Tables[0].Rows[customerSelected];

    frmCustomerDetails.getCustomerDetails(drCustomerWanted);
    frmCustomerDetails.ShowDialog();

    this.Close();
}
```

Run the program. Check that customers can be selected and their details are displayed correctly.

Double click the **'close'** button and add the command **Close()** to close the form.

Double click the **'delete record'** button, and add code to display a confirm message for the user.

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

private void btnDelete_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Really delete?", "Confirm delete",
        MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
    }
}
```

Add lines of code to delete the database record if the user clicks to confirm deletion.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Really delete?", "Confirm delete",
        MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        int CustomerID = Convert.ToInt16(lblCustomerID.Text);
        SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
            AttachDbFilename="+databaseLocation + "Integrated Security=True;
            Connect Timeout=30; User Instance=True");
        try
        {
            con.Open();
            SqlCommand cmCustomers = new SqlCommand();
            cmCustomers.Connection = con;
            cmCustomers.CommandType = CommandType.Text;
            cmCustomers.CommandText =
                "DELETE customer WHERE customerID='" + CustomerID + "'";
            cmCustomers.ExecuteNonQuery();
            con.Close();
            this.Close();
        }
        catch
        {
            MessageBox.Show("File error");
        }
    }
}
```

Double click the *'save changes'* button to create an event method. Add code to collect data from the textBoxes and comboBoxes and store it as variables, ready for updating the database record.

```
private void btnSave_Click(object sender, EventArgs e)
{
    int CustomerID = Convert.ToInt16(lblCustomerID.Text);
    string Surname = txtSurname.Text;
    string Forename = txtForename.Text;
    double Maxprice = Convert.ToDouble(txtMaxprice.Text);
    int Minbedrooms = Convert.ToInt16(txtMinbeds.Text);
    int Typewanted = comboBox1.SelectedIndex;
    int Locationwanted = comboBox2.SelectedIndex;
    int Landwanted = comboBox3.SelectedIndex;

    SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
        AttachDbFilename=" + databaseLocation + "Integrated Security=True;
        Connect Timeout=30; User Instance=True");
}
```

We will complete the *btnSave_Click()* method by adding code for an **UPDATE** command in SQL.

```

SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
AttachDbFilename="+ databaseLocation + "Integrated Security=True;
Connect Timeout=30; User Instance=True");

try
{
    con.Open();
    SqlCommand cmCustomer = new SqlCommand();
    cmCustomer.Connection = con;
    cmCustomer.CommandType = CommandType.Text;
    cmCustomer.CommandText = "UPDATE customer SET surname='" + Surname
        + "', forename='" + Forename + "', maxprice='" + Maxprice
        + "', Minbeds='" + Minbedrooms + "', typeWanted='" + Typewanted
        + "', locationWanted='" + Locationwanted + "', landWanted='"
        + Landwanted + "' WHERE customerID='" + CustomerID + "'";
    cmCustomer.ExecuteNonQuery();
    con.Close();
    this.Close();
}
catch
{
    MessageBox.Show("File error");
}
}

```

One final option provided by our program is to search for suitable properties for each customer. Begin by adding a new **Windows Form** and give this the name '*SearchProperties*'.

Add labels and textBoxes to the form to display the **customer's name**, **maximum price** they wish to pay, **minimum number of bedrooms** required, and any particular requirements for **house type**, **location** or **land**. Name the text boxes as *txtCustomer*, *txtMaxprice*, *txtMinbeds*, *txtTypewanted*, *txtLocationwanted* and *txtLandwanted*.

Below the text boxes, insert a **DataGridView** component. At present, this will appear as an empty grey rectangle.

The screenshot shows a Windows Form with the following layout:

- Label: Customer, followed by a text box.
- Label: Maximum price £, followed by a text box.
- Label: Minimum bedrooms, followed by a text box.
- Label: House type wanted, followed by a text box.
- Label: Location wanted, followed by a text box.
- Label: Land wanted, followed by a text box.
- Below the text boxes, the text "SUITABLE PROPERTIES" is centered.
- At the bottom of the form, there is a large, empty grey rectangle representing a DataGridView component.

Change to the **SearchProperties** program code screen. Add the **'using SqlClient'** directive, and the database location. Create an empty method called **propertySearch()**.

```
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
namespace estateAgent
{
    public partial class SearchProperties : Form
    {
        string databaseLocation = "C:\\C#\\estateAgent.mdf;";

        public SearchProperties()
        {
            InitializeComponent();
        }

        public void propertySearch(string customerName, double maxprice,
            int minbeds, int typewanted, int locationwanted, int landwanted)
        {
        }
    }
}
```

Return to the **CustomerDetails** form and double click the **'search properties'** button to create an event method. Add code which will collect the necessary information about the customer's requirements, then transfer this to the SearchProperties form.

```
private void btnSearch_Click(object sender, EventArgs e)
{
    string CustomerName = txtForename.Text + " " + txtSurname.Text;
    double Maxprice = Convert.ToDouble(txtMaxprice.Text);
    int Minbedrooms = Convert.ToInt16(txtMinbeds.Text);
    int Typewanted = comboBox1.SelectedIndex;
    int Locationwanted = comboBox2.SelectedIndex;
    int Landwanted = comboBox3.SelectedIndex;

    SearchProperties frmSearchProperties = new SearchProperties();

    frmSearchProperties.propertySearch(CustomerName, Maxprice, Minbedrooms,
        Typewanted, Locationwanted, Landwanted);
    frmSearchProperties.ShowDialog();

    this.Close();
}
```

Go to the **SearchProperties** form and find the **propertySearch()** method. Add code to display the customer requirements in the textBoxes.

```
public void propertySearch(string customerName, double maxprice,
    int minbeds, int typewanted, int locationwanted, int landwanted)
{
    txtCustomer.Text = customerName;
    txtMaxprice.Text = String.Format("{0:0,0}", maxprice);
    txtMinbeds.Text = Convert.ToString(minbeds);
    string s = "";
    switch (typewanted)
    {
        case 0: s = "NO PREFERENCE"; break;
        case 1: s = "Detached house (1)"; break;
        case 2: s = "Semi-detached house (2)"; break;
        case 3: s = "Bungalow (3)"; break;
        case 4: s = "Terraced house (4)"; break;
    }
    txtTypewanted.Text = s;
    switch (locationwanted)
    {
        case 0: s = "NO PREFERENCE"; break;
        case 1: s = "Town (1)"; break;
        case 2: s = "Village (2)"; break;
        case 3: s = "Country (3)"; break;
    }
    txtLocationwanted.Text = s;
    switch (landwanted)
    {
        case 0: s = "NO PREFERENCE"; break;
        case 1: s = "Small garden (1)"; break;
        case 2: s = "Large garden (2)"; break;
        case 3: s = "Agricultural land (3)"; break;
    }
    txtLandwanted.Text = s;
}
}
```

Run the program, select a customer, then click to search for suitable properties. The SearchProperties form should open, and the customer's requirements should be displayed:

The screenshot shows a Windows application window titled "SearchProperties". On the left, there is a sidebar with a list of customer records. The selected customer is "Customer ID 2" with the following details: Surname: Humphries, Forename: Stuart, Maximum Price £: 400,000, Minimum Bedrooms: 2, House type wanted: NO PREFERE, Location wanted: village, and Land wanted: large garden. The main area of the window displays the search results for this customer. It shows the Customer name as "Stuart Humphries", Maximum price £ as "400,000", Minimum bedrooms as "2", House type wanted as "NO PREFERENCE", Location wanted as "Village (2)", and Land wanted as "Large garden (2)". Below these fields, the text "SUITABLE PROPERTIES" is displayed, followed by a large grey rectangular area, likely representing a list of search results.

We will now set up an **SQL query** to identify suitable properties for the customer.

Continue the **propertySearch()** method by creating a **dataSet** to hold the results of the query. Add a connection to the database, and set up the structure for a **TRY ... CATCH** block.

```

switch (landwanted)
{
    case 0: s = "NO PREFERENCE"; break;
    case 1: s = "Small garden (1)"; break;
    case 2: s = "Large garden (2)"; break;
    case 3: s = "Agricultural land (3)"; break;
}
txtLandwanted.Text = s;

DataSet dsProperty = new DataSet();

SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS;
    AttachDbFilename="+databaseLocation + "Integrated Security=True;
    Connect Timeout=30; User Instance=True");
try
{
}
catch
{
    MessageBox.Show("File error");
}
}

```

Code can then be added to create the query:

```

try
{
    con.Open();
    SqlCommand cmProperty = new SqlCommand();
    cmProperty.Connection = con;
    cmProperty.CommandType = CommandType.Text;
    string query = "SELECT * FROM house WHERE price<=" + maxprice
        + " AND bedrooms>=" + minbeds;
    if (typewanted > 0)
    {
        query = query + " AND propertyType =" + typewanted;
    }
    if (locationwanted > 0)
    {
        query = query + " AND location =" + locationwanted;
    }
    if (landwanted > 0)
    {
        query = query + " AND land =" + landwanted;
    }
    cmProperty.CommandText = query;

    SqlDataAdapter daProperty = new SqlDataAdapter(cmProperty);
    daProperty.Fill(dsProperty);
    con.Close();
}
catch
{
    MessageBox.Show("File error");
}

```

Look carefully at the way in which the query is built up.

- We begin by asking the computer to select only the records for which the **house price** is less than or equal to the **maximum price** the customer wishes to pay.
- We add a condition that the number of **bedrooms** must be greater than or equal to the **minimum number of bedrooms** required.
- If the customer has indicated a requirement for a particular type of house then we will select only the records for this house type. If the customer has expressed 'NO PREFERENCE' by entering a **typewanted** code of zero, then we will not add this extra requirement to the query.
- Similarly, we will add extra requirements for the house location and land included, unless the customer has expressed 'NO PREFERENCE' for these factors.

Once the query has been constructed and run by the program, suitable house records will be selected from the database and transferred to the **dsProperty** dataset.

The final step is to display the results of the query in the dataGrid:

```

daProperty.Fill(dsProperty);
con.Close();

dataGridView1.DataSource = dsProperty.Tables[0];

DataGridViewColumn column = dataGridView1.Columns[1];
column.Width = 160;
}
catch
{
    MessageBox.Show("File error");
}

```

Run the complete program, and check that suitable properties are selected to meet the customers' requirements:

The screenshot shows a Windows application window with a search form and a data grid. The search form includes the following fields:

- Customer: Elisabeth Edwards
- Maximum price £: 500,000
- Minimum bedrooms: 2
- House type wanted: Detached house (1)
- Location wanted: NO PREFERENCE
- Land wanted: NO PREFERENCE

Below the search form, the text "SUITABLE PROPERTIES" is displayed above a data grid. The data grid has the following columns: houseID, address1, address2, price, bedrooms, propertyType, and location. The data rows are:

	houseID	address1	address2	price	bedrooms	propertyType	location
▶	3	Part Mawr Farnhouse	Bala	464000	4	1	3
	4	The Old Chapel	Tanygrisiau	258000	4	1	2
*							

Carry out final testing of all the program options, including options to edit and delete customer records. Add **Close()** commands to 'Close' buttons where necessary.