MIS 333K Class 14: Seed Method and Search

# Vocabulary

LINQ

Object Initializer

Seed method

Task List (Tool in Visual Studio)

Seed Method

## Migrations Folder Content

When you enable migrations in your project, entity framework creates a migration folder with two files.

### A file named something like 201610032137035\_InitialSetup.cs and contains the code that EF will execute to make your database match your model. An additional file with a similar name and similar contents is created every time you add a migration to your project.

### A file named Configuration.cs contains all of the configuration code for migrations. This file is only created once – it will not be changed by subsequent migrations. You have to change this yourself. This file contains a class called Configuration. Inside that class, there is a method called “Seed.” This method is called every time you run the update-database command in the Package Manager Console window. This method allows you to create seed data for your database without having to manually add it like in HW4.

## Using the Seed Method to Add Data

To get “seed” data in the database, you need to add code to the seed method to create objects that will then be added to your database. Use the following steps:

### Instantiate a new object of the type you are trying to add. Either one of the following ways will work:

#### You can do it the “long” way:

Frequency f1 = new Frequency();

f1.Name = "Daily";

#### OR you can use an Object Initializer to shorten things up.

Frequency f1 = new Frequency(){ Name = "Daily" };

### Add the new object to the database using the .AddOrUpdate() method. NOTE: This method is specific to EF Migrations, so you will need to add a using statement to include the System.Data.Entity.Migrations namespace.

db.Frequencies.AddOrUpdate(f => f.Name, f1)

### Use db.SaveChanges() frequently. This is what actually executes the updates on your database. It is the equivalent of hitting "Save" in Word or Excel. If your updates aren’t being reflected in your tables, see if adding SaveChanges() helps.

## ORDER MATTERS WHEN SEEDING DATA

Pay attention to parent/child relationships when seeding data in your database. You will need to add the parent class data in first. You may also need to add data to scalar properties on both sides of a relationship before you can populate the navigational properties. Slow down and think a little here – a little logic will go a long way in helping you to determine the order to add data.

# Passing Parameters in HTTP Request

## Set up your view to be able to display search.

Since we aren’t adding, editing, or deleting any data in the database with search, we ONLY NEED A GET action method. We need to modify the view on our form to generate a get request and call a different action method for this special case. Adding the line of code below will change the action of the button to generate a GET request and call the “SearchResults” method (even though this code is on the “DetailedSearch” view). The bulk of your code will go inside of curly braces { } that follow this statement

@using (Html.BeginForm("SearchResults", "Home", FormMethod.Get))

{

...code for form controls (text boxes, drop-downs, etc.) goes here

}

## Capture input criteria from the user

To execute search, you need to know what the user is searching for. You will add code to your view to capture the user’s desired search criteria. Several different types of controls can capture input from the user. The different options are applicable in different scenarios. Pay attention to the return type of each of the controls. You will need to reference the return type in the method signature for the action method in your controller.

|  |  |  |  |
| --- | --- | --- | --- |
| **Form Control Types** | | | |
| **Input Type** | **Best Used For** | **Return Type** | **Notes** |
| Textbox | Free-form entry (strings, numbers) | String (Convert to other type if needed) |  |
| Radio Button | Pre-defined list with mutually exclusive options (pick only one) | int or enum value | You will create an html element for each button. Give them all the same name to capture the choice. |
| Checkboxes | Pre-defined list with zero, one or many options | Boolean | Each option is a separate element that will return a value. You need to include a second “hidden” element to get the response to include a value of false when the button is unchecked |
| Drop-down list | Dynamically generated list with mutually exclusive options | int | Remember to add code on your controller that populates a SelectList with the options |
| Listbox | Dynamically generated list with zero, one, or many options | int[] | Remember to add code on your controller that populates a MultiSelectList with the options |

To use one of the above controls, you have to change your code in three places

1. On the view – add the html/razor code to display the control on your search page
2. On the controller – add a parameter to your method signature to capture the input as part of your http request
3. On the controller – add the code to check if the user specified a value for the parameter. If the user specified a value, use the value on LINQ query to limit the results shown. See LINQ info below for more details.

See the Search Demo repository on the class GitHub site for working code examples for each of these. <https://github.com/mis333k-fall2016/Fall-2016-Search-Demo.git>

# Search

To execute search in MVC with Entity Framework, you will need LINQ. LINQ stands for **L**anguage **IN**tegrated **Q**uery. LINQ is a powerful tool for querying data both in databases and in lists and other collections. For search, you will be using LINQ to limit the list of records returned to your search results view. LINQ takes a little getting used to after spending so much time with SQL, but there are only a few basic commands that you need.

## Start with the db set with the data you want

var query = from c in db.Customers

select c;

## Add in “where” clauses to limit the data.

### “Where” clause with an exact match

query = query.Where(c => c.Gender == "Male");

### “Where” clause with a keyword

query = query.Where(c => c.FirstName.Contains("Bob"));

### “Where” clause with keyword in multiple fields

query = query.Where(c => c.FirstName.Contains("Bob") || c.LastName.Contains("Bob"));

### “Where” clause for a range

query = query.Where(c => c.AverageSale >= 300);

### “Where” clause with an “AND”

query = query.Where(c => c.AverageSale >= 300 && c.AverageSale <= 1000);

## Apply multiple where clauses.

Each subsequent where clause further filters the data. For example if you applied a where clause that searches for “female” for gender and then applied a second where clause for sales > 300, you will get female customers with sales over 300. Male customers and customers with less than 300 will be filtered out.

## Add an Orderby statement to sort the data

### Orderby with only one sort field:

query = query.OrderBy(c => c.LastName);

### Orderby with two sort fields:

query = query.OrderBy(c => c.LastName).ThenBy(c => c.FirstName);

## Execute the query by calling the .ToList() method

List<Customer> SelectedCustomers = query.ToList();

## Send the newly limited list to the view to be displayed. Note that you may need to specify the view that displays here. The following line of code causes MVC to render the Index view with the newly filtered list of customers.

return View("Index", SelectedCustomers);

# Code Examples (From Search Demo – See GitHub for full working code)

## Controllers 🡪HomeController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using SearchDemo.Models;

namespace SearchDemo.Controllers

{

public enum Classification { All, Freshman, Sophomore, Junior, Senior }

public enum SortOrder { Ascending, Descending }

public class HomeController : Controller

{

// GET: Home

public ActionResult Index()

{

ViewBag.AllMonths = GetAllMonths();

ViewBag.AllDays = GetAllDays();

return View();

}

public ActionResult SearchResults(String SearchString, Classification SelectedClass, Boolean Transfer, int SelectedMonth, int[] SelectedDays, String DesiredGPA, SortOrder SelectedSortOrder)

{

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: Code for textbox searching (textbox contains a string

//SearchString is the string from the first textbox

if (SearchString == null || SearchString == "") //they didn't select anything

{

ViewBag.SearchString = "Search string was null";

}

else //they picked something

{

ViewBag.SearchString = "The search string is " + SearchString;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: Code for radio buttons

//Figure out selected class

switch (SelectedClass)

{

case Classification.Freshman:

ViewBag.SelectedClassification = "The selected classification is Freshman";

break;

case Classification.Sophomore:

ViewBag.SelectedClassfication = "The selected classification is Sophomore";

break;

case Classification.Junior:

ViewBag.SelectedClassfication = "The selected classification is Junior";

break;

case Classification.Senior:

ViewBag.SelectedClassfication = "The selected classification is Senior";

break;

default:

ViewBag.SelectedClassification = "No classification selected";

break;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: Code for the checkbox

if (Transfer == true)

{

ViewBag.TransferSelected = "Transfer selected";

}

else

{

ViewBag.TransferSelected = "Transfer not selected";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: Code for drop-down list

//Selected month is the selected value from the dropdown

if (SelectedMonth == 0) // they chose "all months from the drop-down

{

ViewBag.SelectedMonth = "No month was selected";

}

else //month was chosen

{

List<Month> AllMonths = MonthUtilities.GetMonths();

Month MonthToDisplay = AllMonths.Find(m => m.MonthID == SelectedMonth);

ViewBag.SelectedMonth = "The selected month is " + MonthToDisplay.MonthName;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: Code for listbox (multiselect)

//loop through and display the days

if (SelectedDays == null || SelectedDays.Count() == 0) //nothing was selected

{

ViewBag.SelectedDays = "No days were selected";

}

else //they picked some days

{

//start the string

String strSelectedDays = "The selected day(s) is/are: ";

//get the list of days

List<Weekday> AllDays = DayUtilities.GetAllDays();

foreach(int DayID in SelectedDays)

{

Weekday DayToDisplay = AllDays.Find(d => d.WeekdayID == DayID);

strSelectedDays = strSelectedDays + DayToDisplay.Name + " ";

}

//add the completed string to viewbag

ViewBag.SelectedDays = strSelectedDays;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: Code for textbox with numeric input

//see if they specified something for GPA

if (DesiredGPA != null && DesiredGPA != "")

//make sure string is a valid number

{

Decimal decGPA;

try

{

decGPA = Convert.ToDecimal(DesiredGPA);

}

catch //this code will display when something is wrong

{

//Add a message for the viewbag

ViewBag.Message = DesiredGPA + "is not valid number. Please try again";

//Re-populate dropdown

ViewBag.AllMonths = GetAllMonths();

ViewBag.AllDays = GetAllDays();

//Send user back to home page

return View("Index");

}

//Do some math with this number to prove it's a number

decGPA += 100; //this is a stupid thing to do;

//You wouldn't want to do it in real life.

//I'm just showing you that it is a number

//and not a string.

//Add value to ViewBag

ViewBag.UpdatedGPA = "The updated GPA is " + decGPA.ToString("n2");

}

else //they didn't specify GPA

{

ViewBag.UpdatedGPA = "No GPA was specified";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TODO: More code for radio buttons (must pick one)

//Figure out sort order

if (SelectedSortOrder == SortOrder.Ascending)

{

ViewBag.SelectedSO = "The records should be sorted in ascending order";

}

else

{

ViewBag.SelectedSO = "The records should be sored in descending order";

}

//send all this stuff to the view

return View();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public SelectList GetAllMonths()

{

List<Month> Months = MonthUtilities.GetMonths();

//add a record for all months

Month SelectNone = new Models.Month() { MonthID = 0, MonthName = "All Months" };

Months.Add(SelectNone);

//convert list to select list

SelectList AllMonths = new SelectList(Months.OrderBy(m => m.MonthID), "MonthID", "MonthName");

//return the select list

return AllMonths;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public MultiSelectList GetAllDays()

{

//Get the list from the other class

List<Weekday> Days = DayUtilities.GetAllDays();

//convert list to multiselect list

MultiSelectList AllDays = new SelectList(Days.OrderBy(d => d.WeekdayID), "WeekdayID", "Name");

//return the multiselect list

return AllDays;

}

}

}

## Views 🡪 Home 🡪 Index.cshtml

@using SearchDemo.Controllers

@{

ViewBag.Title = "Detailed Search";

}

<h2>Advanced Search</h2>

@using (Html.BeginForm("SearchResults", "Home", FormMethod.Get))

{

<p class="form-group">

Name: @Html.TextBox("SearchString", null, new { @class = "form-control" }) <br />

<div class="form-group">

<div class="form-check">

<label class="form-check-label">

@Html.RadioButton("SelectedClass", Classification.All, true, new { @class ="form-check-input"}) All

</label>

</div>

<div class="form-check">

<label class="form-check-label">

@Html.RadioButton("SelectedClass", Classification.Freshman, new { @class = "form-check-input" }) Freshman

</label>

</div>

<div class="form-check">

<label class="form-check-label">

@Html.RadioButton("SelectedClass", Classification.Sophomore, new { @class = "form-check-input" }) Sophomore

</label>

</div>

<div class="form-check">

<label class="form-check-label">

@Html.RadioButton("SelectedClass", Classification.Junior, new { @class = "form-check-input" }) Junior

</label>

</div>

<div class="form-check">

<label class="form-check-label">

@Html.RadioButton("SelectedClass", Classification.Senior, new { @class = "form-check-input" }) Senior

</label>

</div>

</div>

<div class="form-group">

@Html.Label("Transfer?", new { @class = "form-check-label"})

@Html.CheckBox("Transfer", true, new { @class = "form-check-input" })

@Html.Hidden("Transfer") @\*needed for submitting false\*@

</div>

@Html.DropDownList("SelectedMonth", (SelectList)ViewBag.AllMonths, new { @class = "form-control" })

<br/><br />

@Html.ListBox("SelectedDays", (MultiSelectList)ViewBag.AllDays, new { @class = "form-control"})

<br /><br />

<div class="form-group">

GPA: @Html.TextBox("DesiredGPA", null, new { @class = "form-control"}) <div class="text-danger">@ViewBag.Message</div>

<label class="form-check-label">

@Html.RadioButton("SelectedSortOrder", SortOrder.Ascending, true, new { @class = "form-check-input"}) Ascending

</label>

<label class="form-check-label">

@Html.RadioButton("SelectedSortOrder", SortOrder.Descending, new { @class = "form-check-input" }) Descending

</label>

</div>

<input type = "submit" value = "Search" class="btn btn-default" />

</p>

}