WEB524

WEB PROGRAMMING ON WINDOWS

WEEK 6 – LECTURE 1

MANY-TO-MANY & SELF-REFERENCING ASSOCIATIONS

Today

- We will look at two new types of associations
  - Many-to-many
  - Self-referencing (to-one and to-many)

Many-to-Many Associations

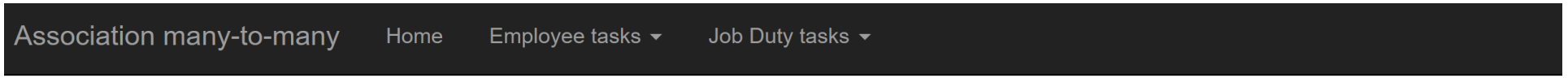
- This pattern is similar to the one-to-many pattern.
- We are almost always working with one object at a time at one end of the association.

AssocManyToMany Code Example

- Download and study the AssocManyToMany code example.
- There is a simple design model consisting of two classes: Employee and JobDuty.
  - These classes have a to-many association with each other.
  - Sample data comes from the School of SDSS.
  - An employees can have multiple job duties. Some of your “professors” are also “program coordinators”, or “researchers”. All of these are job duties.
  - A job duty can be performed by multiple employees. A “student advisor” job duty is performed by Brenda, Peter, Ian, and others in the School of SDSS office.
  - Class diagram follows …


Employee
Class
Properties
Id : int
Name : string
Office : string
Employees : ICollection<Employee>
JobDuties : ICollection<JobDuty>
JobDuty
Class
Properties
Description : string
Id : int
Name : string

Displaying data from a many-to-many association



List of employees

Employee name	Office number	Job duties	
Baker, Brenda	T2076	Student Advisor Admin Support Classroom Technician Chair	Edit job duties | Details
Bhattacharya, Ranjan	S2138	Admin Support Chair Dean	Edit job duties | Details
Bradley, Rose	A4000	Student Advisor Admin Support	Edit job duties | Details
Craigs, Karen	A4023		Edit job duties | Details
Crawford , Patrick	T2094	Professor Curriculum Coordinator	Edit job duties | Details
Czegel, Barb	T2095	Professor Curriculum Coordinator	Edit job duties | Details
Czegel, Les	T2090	Professor	Edit job duties | Details
Daoud, Mariam	T2096	Professor	Edit job duties | Details
Davadpour, Maryam	A4038		Edit job duties | Details

List of Employees

- On the list of employees, the new column “Job duties” was added.
- The view’s model object is a collection of employees.
- Each employee object has a collection of job duties.
- Rendering the list:

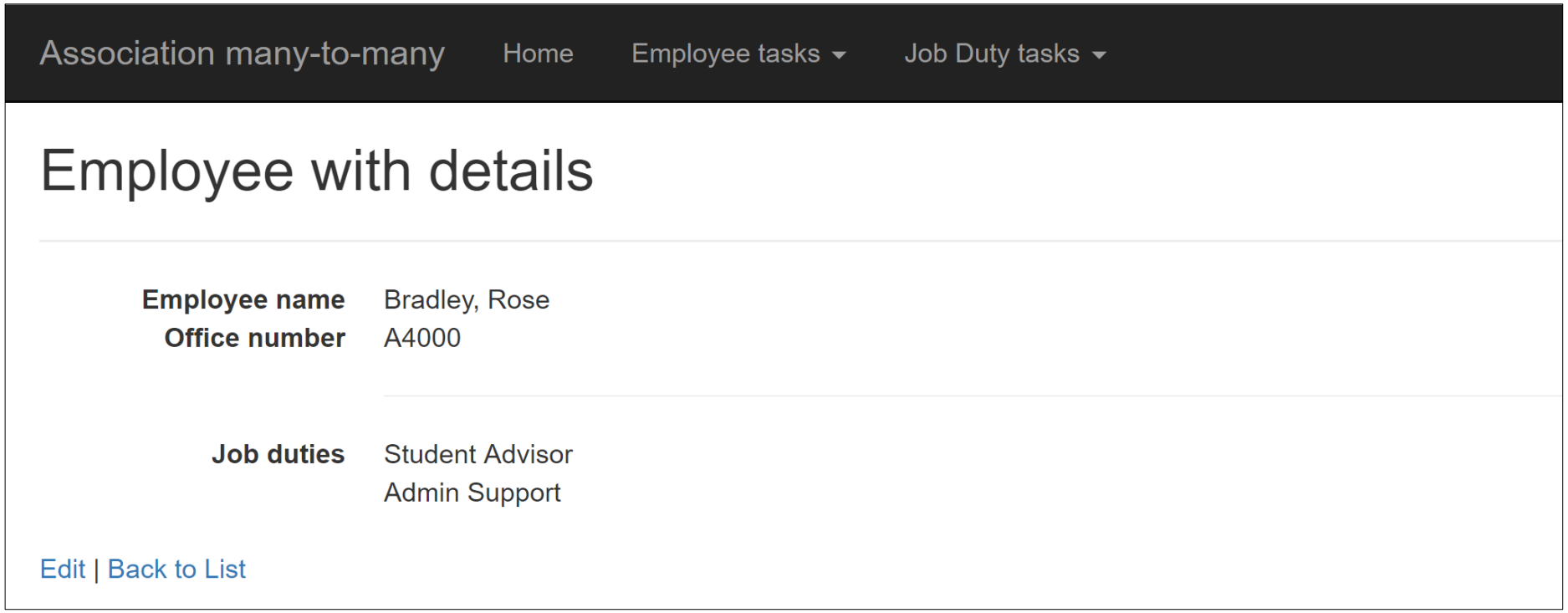
```
<td>
   @if (item.JobDuties.Count() > 0)
   {
      var jd = string.Join("<br />",
                  item.JobDuties.Select(jdn => jdn.Name));

      @Html.Raw(jd)
   }
</td>
```

List of Employees

Notice two things about the previous code example:

1. We are creating a string that includes markup, notice the
 element. If a string includes markup, we MUST use the Raw() HTML Helper to pass it to the view, unencoded.
2. Notice the use of the Select() method on the collection. It enables us to select one specific property from each item in the collection instead of having to work with all properties.

   The result of item.JobDuties.Select() is an array of strings. It is passed to the string.Join() method to create a nicely-formatted list of job duty strings.


Association many-to-many
Home
Employee tasks
Job Duty tasks
Employee with details
Employee name
Bradley, Rose
Office number
A4000
Job duties
Student Advisor
Admin Support
Edit | Back to List

Employee Details

- In the employee details view, the model is an Employee object, it has a collection of job duties.
- Rendering it is easy:

```
<dt>Job duties</dt>
<dd>
   @if (Model.JobDuties.Count() > 0)
   {
      foreach (var jd in Model.JobDuties)
      {
         <span>@jd.Name</span><br />
      }
   }
   else
   {
      <span>(none)</span>
   }
</dd>
```

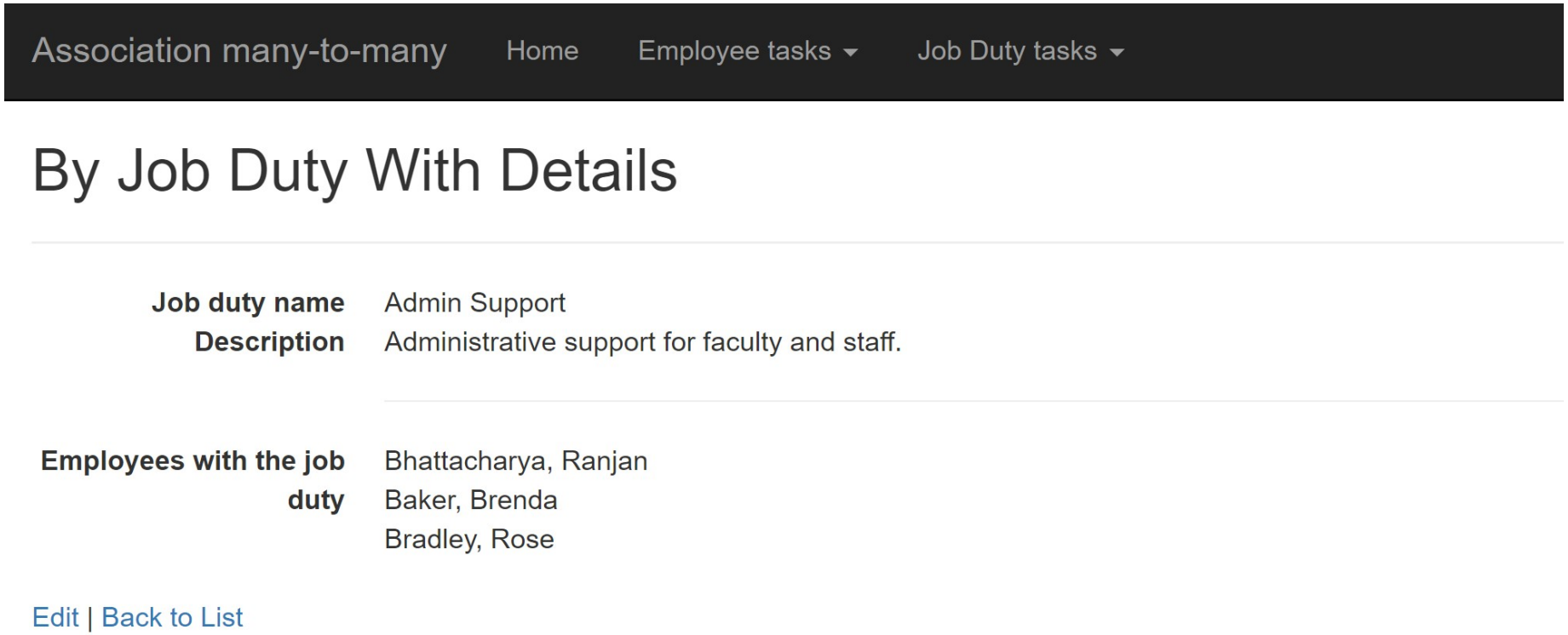
The Job Duty

- Looking at the other end – the job duty – you will notice a similar pattern.
- Following are screen captures of job duty data. Each shows the associated employee data.
- On the list of job duties, the new column “Employees with this job duty” was added.
- The view’s model object is a collection of job duties. Each job duty object has a collection of employees.
- Rendering it is similar to the previous example:

Association many-to-many Home Employee tasks Job Duty tasks

List of job duties

Job duty name	Description	Employees with this job duty	
Admin Support	Administrative support for faculty and staff.	Bhattacharya, Ranjan Baker, Brenda Bradley, Rose	Edit employees | Details
Chair	Leader of the School.	Manton, Mary Lynn Bhattacharya, Ranjan Baker, Brenda	Edit employees | Details
Classroom Technician	Maintains classroom facilities and equipment.	Baker, Brenda	Edit employees | Details
Co-op Coordinator	Foster and maintain employer relationships. Helps students seek work placements.	Harper, Pat	Edit employees | Details
Curriculum Coordinator	Curriculum improvement for an area of the overall curriculum.	Crawford , Patrick Czegel, Barb	Edit employees | Details
Dean	Leader of the Faculty (which consists of many Schools).	Bhattacharya, Ranjan	Edit employees | Details
Professor	Teacher. Course development.	Crawford , Patrick Czegel, Barb Czegel, Les Daoud, Mariam DeJong, Nick Delpisheh, Elnaz Douglas, Brian	Edit employees | Details
Program Coordinator	Program-level curriculum improvement. Student success.	Kubba, Nagham	Edit employees | Details


Association many-to-many
Home
Employee tasks
Job Duty tasks
By Job Duty With Details
Job duty name
Admin Support
Description
Administrative support for faculty and staff.
Employees with the job duty
Bhattacharya, Ranjan
Baker, Brenda
Bradley, Rose
Edit | Back to List

Editing data in a to-many association

- Last week, you learned to work with data in a one-to-many association.
- We focused on selecting and saving a single object – chosen from a collection – as the associated data.
- In other words, for a specific manufacturer, we added a vehicle. Or, for a new vehicle, we configured a specific manufacturer.
- Until now, relationships were treated almost like single entities.

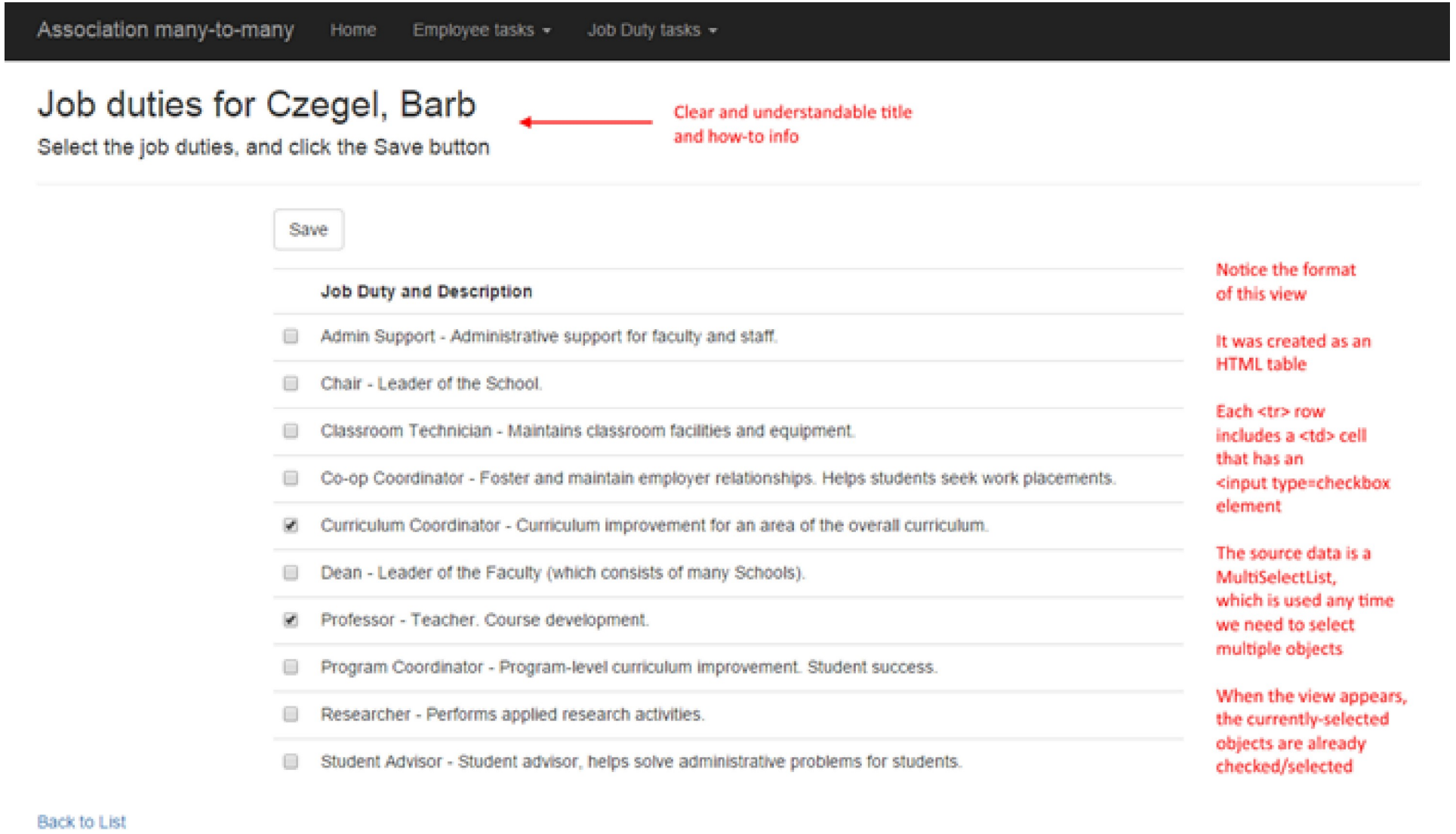
Many-to-many

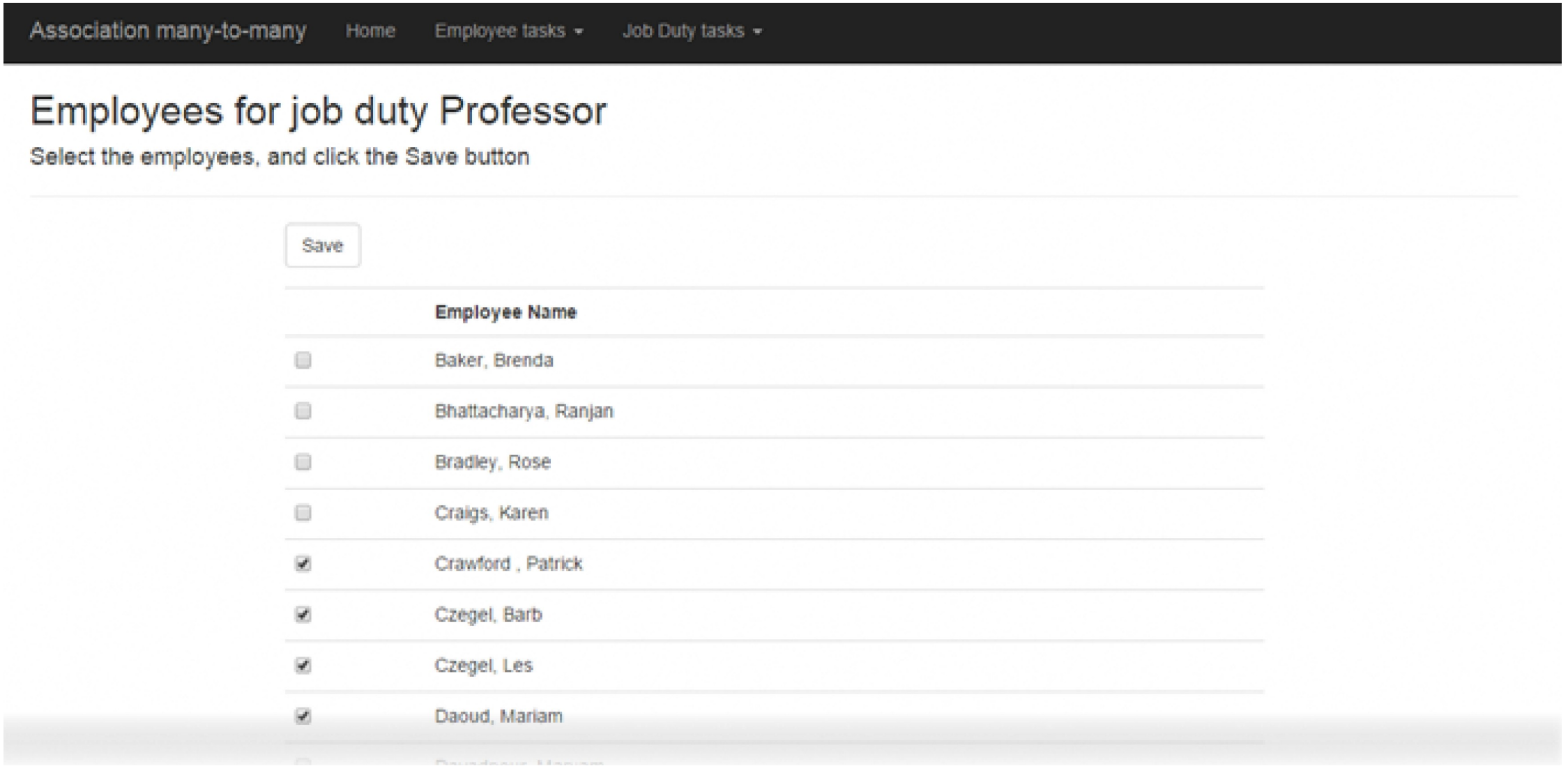
- What's different in a many-to-many association is that often you will want to select and save multiple objects chosen from a collection.
- We need to present a UI to enable that and write code to handle the results.
- Remember the code example, we worked with an Employee object to configure some job duties. (The other way – job duty to employees – would work exactly the same)
- It implements a design and coding approach, for each entity:

Design & Coding Approach

1. Write a "…Base" view model class.
2. Write a "…With…" view model class to add the to-many collection of the other entity.
3. Write a "…Edit…Form" view model class to package the data to be sent to the form.
4. Write an "…Edit…" view model class for the data submitted by the user.
5. In the Manager class, write a method that will process the submitted data.
6. In the Controller class, write a pair of methods (actions) that enable this use case.
7. Generate the view and hand-edit it to add an item-selection element.

- You may end up with something that looks like the following screenshot…


Association many-to-many
Home
Employee tasks
Job Duty tasks
Job duties for Czegel, Barb
Select the job duties, and click the Save button
Clear and understandable title and how-to info
Save
Job Duty and Description
Admin Support - Administrative support for faculty and staff.
Chair - Leader of the School.
Classroom Technician - Maintains classroom facilities and equipment.
Co-op Coordinator - Foster and maintain employer relationships. Helps students seek work placements.
Curriculum Coordinator - Curriculum improvement for an area of the overall curriculum.
Dean - Leader of the Faculty (which consists of many Schools).
Professor - Teacher. Course development.
Program Coordinator - Program-level curriculum improvement. Student success.
Researcher - Performs applied research activities.
Student Advisor - Student advisor, helps solve administrative problems for students.
Back to List
Notice the format of this view
It was created as an HTML table
Each <tr> row includes a <td> cell that has an <input type=checkbox element
The source data is a MultiSelectList, which is used any time we need to select multiple objects
When the view appears, the currently-selected objects are already checked/selected


Association many-to-many
Home
Employee tasks
Job Duty tasks
Employees for job duty Professor
Select the employees, and click the Save button
Save
Employee Name
Baker, Brenda
Bhattacharya, Ranjan
Bradley, Rose
Craigs, Karen
Crawford , Patrick
Czegel, Barb
Czegel, Les
Daoud, Mariam

Write a “...Base” view model class

- This should be very familiar at this point.
- Include the properties that you will display and allow editing of.

Write a “…With…” view model class

- Write a “…With…” view model class to add the “to-many” collection of the other entity.
- Add a collection navigation property to hold the associated objects.
- In the code example, the class is named **EmployeeWithJobDutiesViewModel**
- It includes this:

```
public IEnumerable<JobDutyBaseViewModel> JobDuties { get; set; }
```

Write a “…Edit…Form” view model class

- Write a “…Edit…Form” view model class to package the data to be sent to the form.
- This class needs a select list, specifically a **MultiSelectList**. Its data will be used to render the item-selection elements in the HTML form.

```
// Multiple select requires a MultiSelectList object
[Display(Name = "Job duties")]
public MultiSelectList JobDutyList { get; set; }
```

Write an "...Edit..." view model class

- Write an "...Edit..." view model class for the data submitted by the user.
- This class needs only a few properties:
  - The Employee object identifier
  - Optionally, if you allowed editing of any other property, include that property
  - A collection of int identifiers, for the multiple selected objects

```
public int Id { get; set; }

// Incoming collection of selected job duty identifiers
public IEnumerable<int> JobDutyIds { get; set; }
```

Manager Class, Process Data

- In the Manager class, write a method that will process the submitted data.

- When you fetch the Employee object from the data store, you MUST include the associated data.

- The fetched Employee object will already have an existing collection of job duties. The incoming data has a new collection of job duty identifiers. What should we do about that?

Manager Class, Process Data

- One approach is to clear the existing collection of job duties then for each new job duty identifier, fetch the job duty object and add it to the employee's collection of job duties.
- This approach is used in the code example:

```
o.JobDuties.Clear();

foreach (var item in newItem.JobDutyIds)
{
  var a = ds.JobDuties.Find(item);
  o.JobDuties.Add(a);
}
```

Controller Class, Method Pair

- In the Controller class, write the pair of methods that enable this use case.
- The pair of methods will be familiar, except for the configuration of the select list object.
- **New this week:** When we need to edit an employee's collection of job duties, we MUST pay attention to the already-selected items and send that to the select list object. This allows us to show the already-selected items in the view.
- Both **SelectList()** and **MultiSelectList()** have a constructor parameter to hold the selected value(s). In the code example we use the MultiSelectList.
- When we make one of these objects, we need TWO collections:

Controller Class, Two Collections

1. The collection of items that will be rendered in the user interface.
2. A collection of currently (previously) selected values.

- The manager object already has a method to deliver #1, the collection of all job duties.

- Where do we get #2?

Currently Selected Values

- We have already fetched the Employee object that we want to edit, it has the collection we need!
- From that collection of JobDutyBaseViewModel objects, we need only the identifier, so we can use the Select() method again.
- Then, we can build the MultiSelectList object:

Building the MultiSelectList Object

```
var selectedVals = o.JobDuties.Select(jd => jd.Id);

// For clarity, use the named parameter feature of C#
form.JobDutyList = new MultiSelectList (
     items: m.JobDutyGetAll(),
     dataValueField: "Id",
     dataTextField: "FullName",
     selectedValues: selectedVals
);
```

Generate the View

- Generate the view and hand-edit it to add an item-selection element.
- In the code example we are using an HTML table to render the items. You can use a checkbox group or multiple-select listbox if you want.
- **Important:** We MUST render the "checked" attribute for the <input type=checkbox /> element.
- When we use the MultiSelectList constructor (from earlier) with the selectedValues argument, it builds a boolean "Selected" property for each item in the select list object.
- We can render it like this:

```
<input type="checkbox" name="JobDutyIds" value="@item.Value“
checked="@item.Selected" />
```

Self Referencing (to-one)

Self-Referencing Association (to-one)

- Download and study the AssocSelf code example.
  - It uses the music business as its problem domain.
  - Some entities can have “self” associations.
  - The example web app’s “Employee” entity has two self-referencing associations:
    - To-one, used to indicate the supervisor (or boss/manager)
    - To-many, used to indicate the collection of direct-reports
  - This scenario is quite common. For example, if we consider a generic “Person” class, we can have properties for “Mother” and “Father”.
    - Each will point to another existing Person object.
    - Each can also have a “Children” collection property with pointers to a collection of existing Person objects.

Coding the Design Model Class (to-one)

- We MUST follow these rules, for the "to-one" association:
  - TWO properties are required.
  - One property is a nullable int – it will hold the identifier of the associated object.
  - The other property is a reference to the associated object (i.e. its type is the class type)
- When you update this associated object, you must update BOTH properties.
- Here's what the Employee class in the code example looks like:

```
public int? ReportsTo { get; set; }
public Employee Employee2 { get; set; }
```

Note: "Employee2" is a terrible name for this property. Your teacher did not name this property (or another terribly-named property, "Employee1"). For your own projects, do NOT use names like these.

Coding the Design Model Class (to-many)

- When coding the design model class, the “to-many” association looks like any other association you’ve already coded. You can continue to use previously-learned design and coding techniques.
- Here’s what the Employee class in the code example looks like:

```
public virtual ICollection<Employee> Employee1 { get; set; }
```

Note: Do NOT use “virtual” in your own projects. Also, use a better name for the property in your own projects.

Highlights from the “edit supervisor” example

- This section features some highlights from the AssocSelf code example.
- When you run the web app, it shows a list of employees.

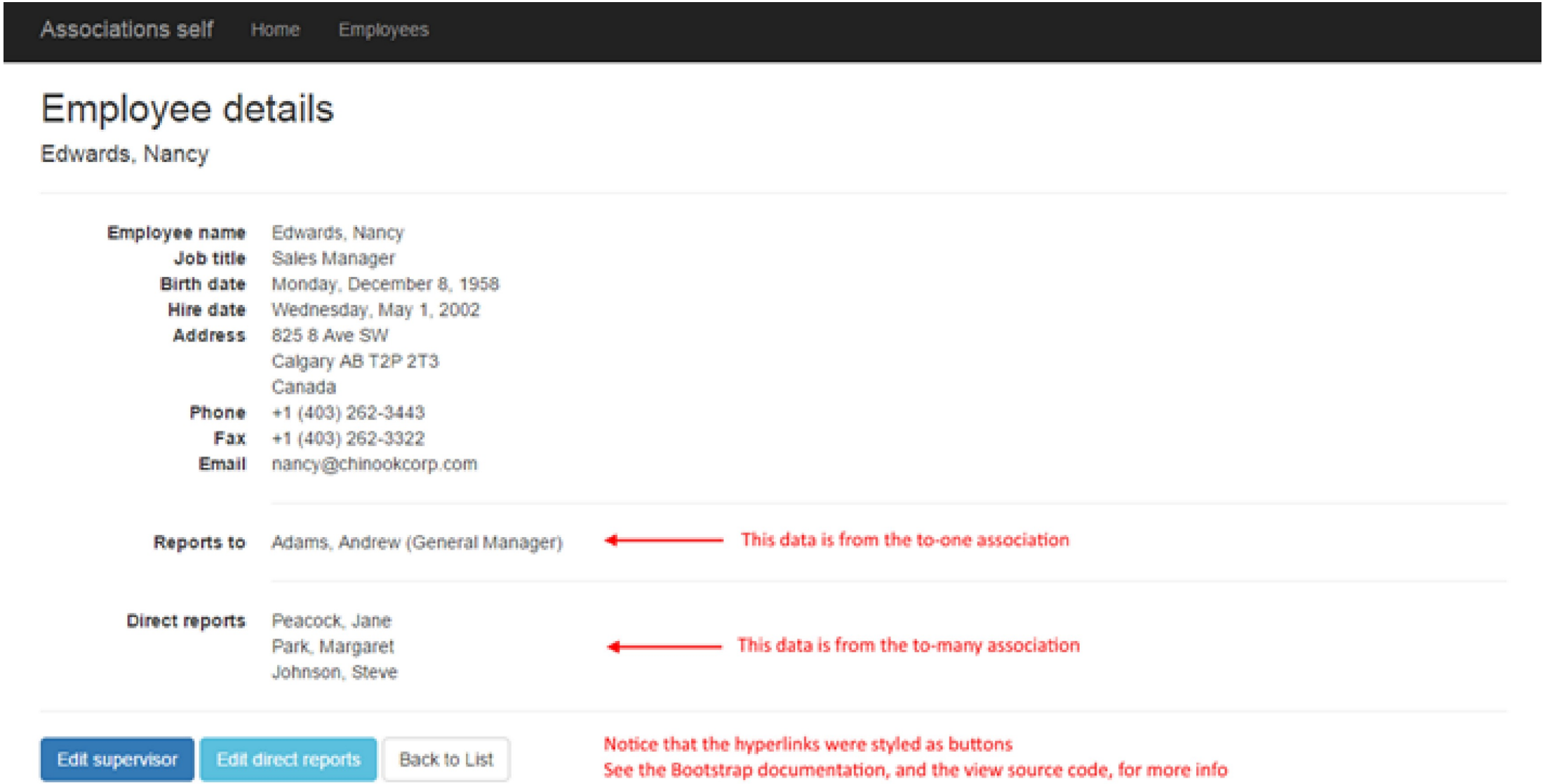
Associations self Home Employees

Employee list

Employee name	Dates	Address	Numbers	Email	Reports to	
Adams, Andrew General Manager	Birth: 1962-02-18 Hire: 2002-08-14	11120 Jasper Ave NW Edmonton AB T5K 2N1 Canada	T: +1 (780) 428-9482 F: +1 (780) 428-3457	andrew@chinookcorp.com	Callahan, Laura	Details
Callahan, Laura IT Staff	Birth: 1968-01-09 Hire: 2004-03-04	923 7 ST NW Lethbridge AB T1H 1Y8 Canada	T: +1 (403) 467-3351 F: +1 (403) 467-8772	laura@chinookcorp.com	Mitchell, Michael	Details
Edwards, Nancy Sales Manager	Birth: 1958-12-08 Hire: 2002-05-01	825 8 Ave SW Calgary AB T2P 2T3 Canada	T: +1 (403) 262-3443 F: +1 (403) 262-3322	nancy@chinookcorp.com	Adams, Andrew	Details
Johnson, Steve Sales Support Agent	Birth: 1965-03-03 Hire: 2003-10-17	7727B 41 Ave Calgary AB T3B 1Y7 Canada	T: 1 (780) 836-9987 F: 1 (780) 836-9543	steve@chinookcorp.com	Adams, Andrew	Details
King, Robert IT Staff	Birth: 1970-05-29 Hire: 2004-01-02	590 Columbia Boulevard West Lethbridge AB T1K 5N8 Canada	T: +1 (403) 456-9986 F: +1 (403) 456-8485	robert@chinookcorp.com	Mitchell, Michael	Details
Mitchell, Michael IT Manager	Birth: 1973-07-01 Hire: 2003-10-17	5827 Bowness Road NW Calgary AB T3B 0C5 Canada	T: +1 (403) 246-9887 F: +1 (403) 246-9899	michael@chinookcorp.com	Adams, Andrew	Details

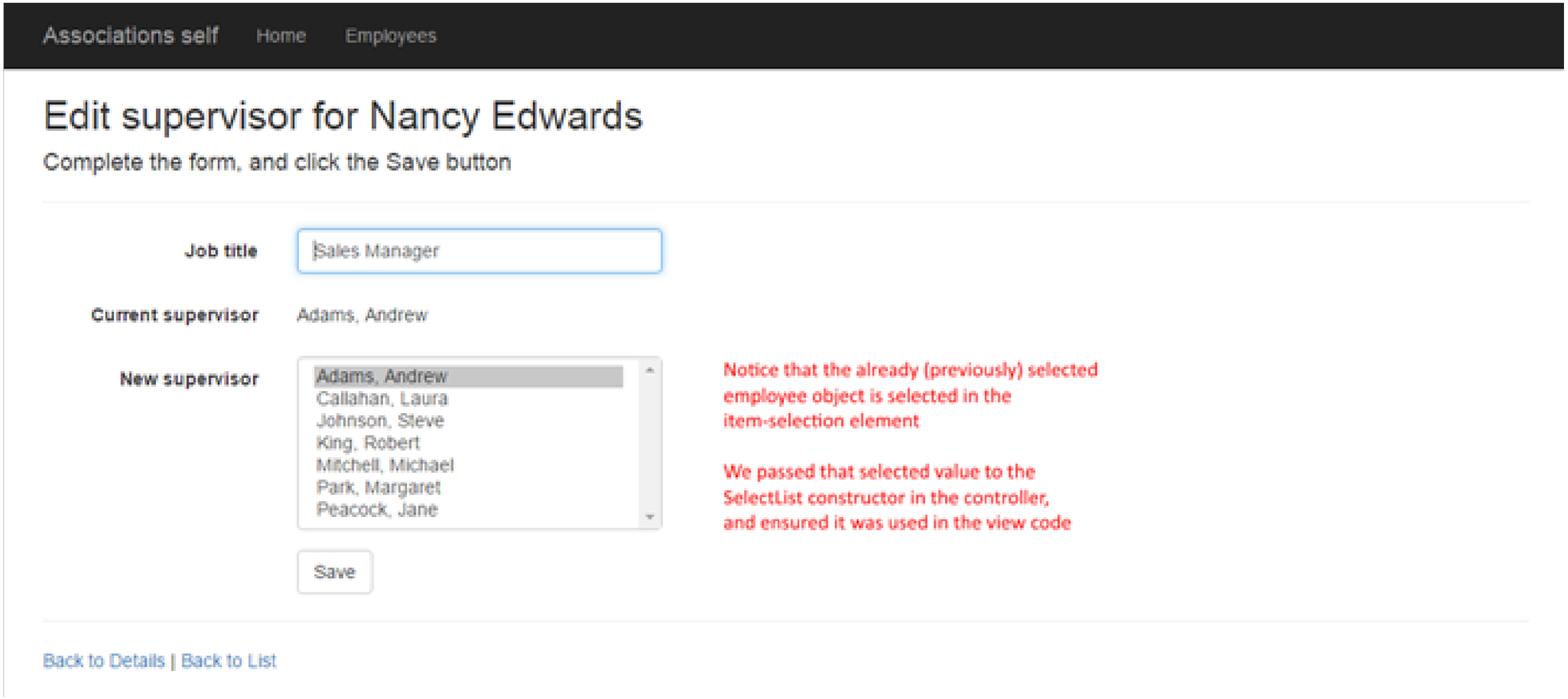
Employees Index

- The previous image shows an edited version of the scaffolded view.
  - The scaffolded view had too many columns, some were removed.
  - A copy of the original scaffolded view has been supplied, it is called “IndexOriginal.cshtml”.
- Notice that the dates were formatted nicely. Study the view code.
- We had to do an additional check *(item.BirthDate.HasValue)* because it was configured as a nullable date, so we have to guard against null values.
- Notice the new “Reports to” column. A collection of employees “with details” was passed to the view.
- The “Details” button is an <a> element/link that has been styled with Bootstrap classes. Following that link will show details for a selected employee.


Associations self
Home
Employees
Employee details
Edwards, Nancy
Employee name
Edwards, Nancy
Job title
Sales Manager
Birth date
Monday, December 8, 1958
Hire date
Wednesday, May 1, 2002
Address
825 8 Ave SW
Calgary AB T2P 2T3
Canada
Phone
+1 (403) 262-3443
Fax
+1 (403) 262-3322
Email
nancy@chinookcorp.com
Reports to
Adams, Andrew (General Manager)
This data is from the to-one association
Direct reports
Peacock, Jane
Park, Margaret
Johnson, Steve
This data is from the to-many association
Edit supervisor
Edit direct reports
Back to List
Notice that the hyperlinks were styled as buttons
See the Bootstrap documentation, and the view source code, for more info

Employee Details

- Study the “Details.cshtml” view code.
- It repeats some of the code from the list-of-employees view however it adds a collection of “direct reports” (employees who report to this employee).
- At this point in time, you can:
  1. Edit the employee’s supervisor, or
  2. Edit the employee’s direct reports
- Let’s look at editing the supervisor first.


Associations self
Home
Employees
Edit supervisor for Nancy Edwards
Complete the form, and click the Save button
Job title
Sales Manager
Current supervisor
Adams, Andrew
New supervisor
Adams, Andrew
Callahan, Laura
Johnson, Steve
King, Robert
Mitchell, Michael
Park, Margaret
Peacock, Jane
Save
Notice that the already (previously) selected employee object is selected in the item-selection element
We passed that selected value to the SelectList constructor in the controller, and ensured it was used in the view code
Back to Details | Back to List

Constructing “Edit the Supervisor”

1. Write the “…Form” view model class (that will be used to build the HTML Form).
2. Write the “…Edit…” view model class for the submitted data from the user.
3. Write the Manager class method to process the submitted data.
4. Write the Controller class method pair.
5. Generate and customize the view.

Write the "...Form" view model class

- This class includes properties to help identify the object, when displayed in the browser.
- Since we're allowing a change in supervisor it's possible that the job title will also change, therefore, we will permit the user to edit the job title.
- For convenience, we will also show/display the employee's current supervisor. Why?
  - When the form loads in the browser, it will show the current supervisor however when the user clicks a different name on the list, the context of the original supervisor is lost.
  - This is simply for informational purposes only so the user does not forget the original selection.
- The class will need a SelectList property to hold the items for an item-selection element on the HTML Form.

Write the "...Edit..." view model class

- This class has the properties:
  - The object identifier
  - The edited job title
  - The identifier of the employee who will be the supervisor

Manager Class

- Use the passed-in object identifier to fetch the Employee that we must update.
- We must attempt to fetch the associated item – the Employee object for the newly-selected supervisor.
- **New:** We continue processing only if BOTH objects exist.
  - If we can continue then we can update values using the SetValues() method as we have done before in "edit existing" use cases.
  - Remember that this method does NOT update object identifiers or navigation properties.
  - Next, we update the navigation properties. There's only one, the supervisor is "Employee2". *Yes, a terrible name for a property*.
  - We MUST also update "ReportsTo" property with the identifier of the supervisor object.

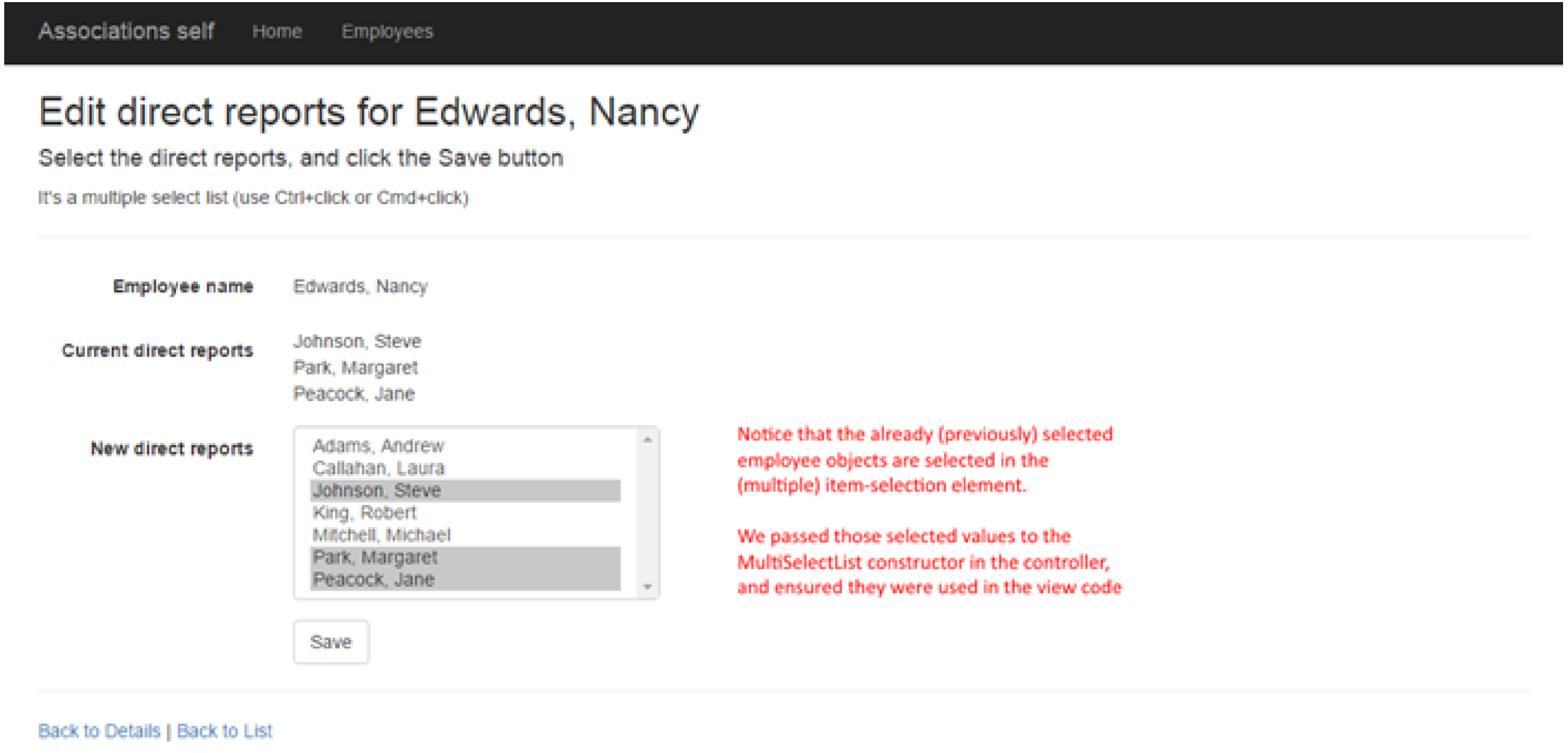
Write the Controller class method pair

- In the GET method, which prepares the "...Form" view model object, we must think about and implement the logic carefully.
  - First, we must fetch a collection of employees so that the employee being edited can select a new supervisor.
  - The employee being edited must NOT be on the list - we must remove it. There's additional logic to do that.
  - Next, we must determine which employee is the current supervisor. We will render that in the user interface and use its identifier when we build the select list object.
  - Finally, we build the SelectList object. Use the employee collection that does NOT include the employee being edited.
- The POST method, which handles the data submitted by the user looks similar to the ones you have written in the past.

Generate the View

- The scaffolder will generate editable textboxes for the name properties but they must not be edited.
- It will not generate an item-selection element.
- We must hand-edit the view to match the earlier screenshot.
- Notice the use of a <p> element with a class to display the name of the current supervisor.

Highlights from the “edit direct reports” example


Associations self
Home
Employees
Edit direct reports for Edwards, Nancy
Select the direct reports, and click the Save button
It's a multiple select list (use Ctrl+click or Cmd+click)
Employee name
Edwards, Nancy
Current direct reports
Johnson, Steve
Park, Margaret
Peacock, Jane
New direct reports
Adams, Andrew
Callahan, Laura
Johnson, Steve
King, Robert
Mitchell, Michael
Park, Margaret
Peacock, Jane
Save
Notice that the already (previously) selected employee objects are selected in the (multiple) item-selection element.
We passed those selected values to the MultiSelectList constructor in the controller, and ensured they were used in the view code
Back to Details | Back to List

Constructing the View

1. Write the "…Form" view model class (that will be used to build the HTML Form).
2. Write the "…Edit…" view model class for the submitted data from the user.
3. Write the Manager class method to process the submitted data.
4. Write the Controller class method pair.
5. Generate and customize the view.

Write the "...Form" view model class

- Similar to the design approach from earlier, we need properties for display and for editing.
- As a convenience, we can have a property, DirectReports, that holds the names of the employees who currently report directly to the employee being edited.
- The select list property is of type MultiSelectList.

Write the "...Edit..." view model class

- This is a very simple class.
- It has:
  - the employee being edited object identifier
  - an int collection for the selected employees

Manager Class

- The design and coding approach will be the same as discussed earlier (in the many-to-many section).
- We must first clear the existing direct reports and then add in the new direct reports.

Controller Method Pair

- For the GET method, the approach will be the same as discussed in the “edit supervisor” example.
- A couple of things are different, however:
  - We get a collection of the existing direct report object identifiers.
  - This collection is used to build the MultiSelectList object.
  - We send along a collection of existing direct report objects (EmployeeBaseViewModel) that will display in the user interface.
  - We build a MultiSelectList object.
- For the POST method, it uses the same familiar pattern as before.

Generate the View

- The scaffolder will generate editable textboxes for the name properties that must not be edited.
- It will not generate an item-selection element.
- We must hand-edit the view to match the screenshot from earlier.

Summary

- Working with many-to-many and self-referencing associations is similar to previous associations.
- Constructing the user interface correctly is a challenge. All programmers, beginners and experienced, must follow a checklist in a disciplined manner because there are simply too many implementation details.
- Be careful and take your time.
- Write a checklist.
- Work incrementally and test frequently.