# Data Models and Databases

Dominic Duggan
Stevens Institute of Technology
Based in part on materials by K. Birman, S. Mitchell

1

## UPLOADING DATA

2

# Image Model

- Model for photographic image

```
public class Image {
   public String Id { get; set; };
   public String Caption { get; set; };
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

3

# Saving Data

```
using Newtonsoft.Json;

public ActionResult Upload(Image image) {

   String jsonData =
       JsonConvert.SerializeObject(image,
                           Formatting.Indented);

   String fileName = ...
   File.WriteAllText(filename, jsonData);
   return View("QuerySuccess", image);
}
```

4

# Saving Data

```
public ActionResult Upload(Image image) {

   try{
      ...
      File.WriteAllText(filename, jsonData);
      return View("QuerySuccess", image);
   } catch (IOException exn) {
      return RedirectToAction ("Error", "Home",
            new { userid=userid, errid="upload" });
   }
}
```

5

# Retrieving Data

```
using Newtonsoft.Json;

public ActionResult Query(string id) {
   ...
   String jsonData = File.ReadAllText(filename);

   Image imageInfo = (Image)
      JsonConvert.DeserializeObject(jsonData);

   return View("QuerySuccess", image);
}
```

6

# Retrieving Data

```
using Newtonsoft.Json;

public ActionResult Query(string id) {
   ...
    if (File.Exists(filename)) {
      String jsonData = File.ReadAllText(filename);

      Image imageInfo = (Image)
         JsonConvert.DeserializeObject(jsonData);

      return View(“QuerySuccess”, image);
   } else {
      ViewBag.message = “Unknown image!”; ...
   }
}
```

7

# Error Reporting

```
public ActionResult Error (String userid, String errid)
{

   ViewBag.Userid = userid;

   if (errid == ”submit") {
      ViewBag.errmsg = "Error trying to submit data!";
   } else {
      ViewBag.errmsg = "Unknown error!";
   }

   return View();
}
```

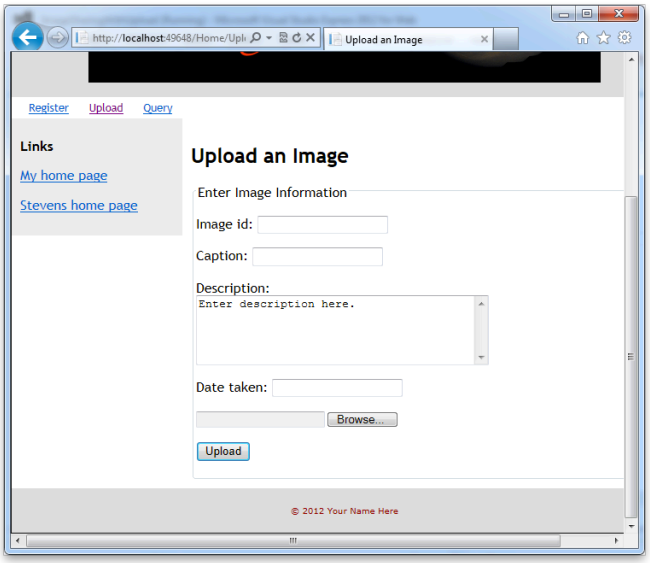
8

# Error Reporting

```
public ActionResult Error (String userid, String errid)
{

   ViewBag.Userid = userid;

   if (errid == ”submit") {
      ViewBag.errmsg = "Error trying to submit data!";
   } else {
      ViewBag.errmsg = "Unknown error!";
   }

   return View();
   LogErrorMessage(userid, errid);
}
```

9

# File Upload

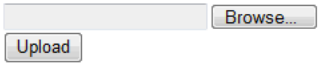



10

# File Upload Form

- Generate the form:

```
<form method="post" asp-action="Upload"
      asp-controller="Home"
      enctype="multipart/form-data" >
    <input type="file" name="ImageFile" />
    <br/>
    <input type="submit" value="Upload"
           name="Submit" id="Submit" />
</form>
```

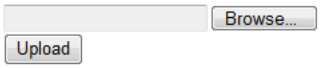



11

# File Upload Form

- Resulting HTML:

```
<form action="/Home/Upload" method="post"
      enctype="multipart/form-data">
   <input type="file" name="ImageFile" />
   <br/>
   <input type="submit" value="Upload"
          name="Submit" id="Submit" />
</form>
```




12

# Processing Upload

```
[HttpPost]
public ActionResult Upload
      (Image image, IFormFile ImageFile) {
   if (ImageFile != null &&
       ImageFile.Length > 0) {
      ...
      // save the file on the server
      ...
   }
   return View();
}
```

13

# Naming Files

```
private readonly IHostingEnvironment hostingEnvironment;
public HomeController(IHostingEnvironment environment) {
   hostingEnvironment = environment;
}

protected string imageDataFile(string id) {
   return Path.combine(
      hostingEnvironment.WebRootPath,
      “data”, “images”, id + “.jpg”);
}

protected string imageInfoFile(string id) {
   return Path.combine(
      hostingEnvironment.WebRootPath,
      “data”, “info”, id + “.js”);
}
```

14

# Processing Upload

```
[HttpPost]
public ActionResult Upload
      (Image image, IFormFile ImageFile) {
   if (ImageFile != null &&
       ImageFile.Length > 0) {
       ImageFile.CopyToAsync(
         imageDataFile(image.id),
         FileMode.Create);
   }
   return View();
}
```

15

# Processing Upload

```
[HttpPost]
public ActionResult Upload (Image image,
                            IFormFile ImageFile) {
   if (ImageFile != null && ImageFile.Length > 0) {
      if (ImageFile.Length > LIMIT) {
         // Report an error on file size
      } else if (ImageFile.ContentType != “image/jpeg”) {
         // Error on content type; but can be faked!
      } else {
         ImageFile.CopyToAsync(
            imageDataFile(image.id), FileMode.Create);
      }
   }
   return View();
}
```

16

# Validating an Image File

- Turn the file into an image

```
System.Drawing.Image img =
   System.Drawing.Image.FromStream
      (ImageFile.OpenReadStream());
if (img.RawFormat.Guid  ==
 System.Drawing.Imaging.ImageFormat.Jpeg.Guid)
{
   ImageFile.CopyToAsync(...);
}
```

17

# Validating an Image File

- Return the type of the image

```
public static string MimeType
         (System.Drawing.Image image)
{
   foreach (ImageCodecInfo codec in
            ImageCodecInfo.GetImageDecoders()) {
      if (codec.FormatID ==
          image.RawFormat.Guid) {
         return codec.MimeType;
      }
   }
   return "image/unknown";
}
```

18

## SESSION STATE

19

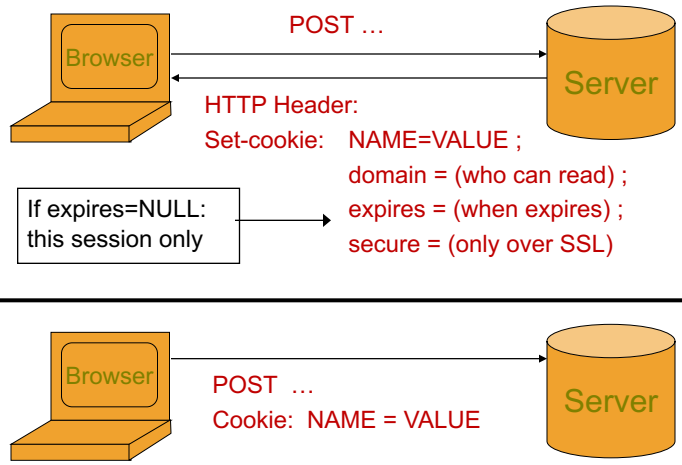
## Session State

- Example: User logs in
  - Credentials?
- Example: User checks out
  - Contents of shopping cart?
- Challenge: Web servers are *stateless*

20

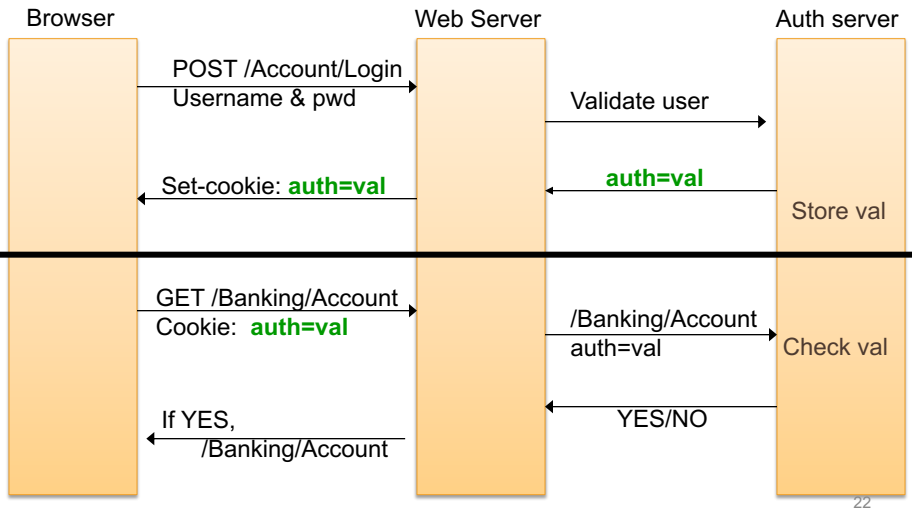
# Cookies

- Used to store state on user's machine




21

# Cookie authentication




22

# Cookies

- Writing a value to a cookie:

```
var options = new CookieOptions() {
   IsEssential = true,
   Expires = DateTime.Now.AddMonths(3)
};
Response.Cookies.Append(
   “cookieName”, ..., options);
```

- Reading the value from a cookie:

```
string cookie =
   Request.Cookies[“cookieName”];
```

23

# Cookies

- Cookies are sent back and forth in plain-text.
- Cookies are stored on the client’s computer.
- Never store sensitive information in cookies.

24

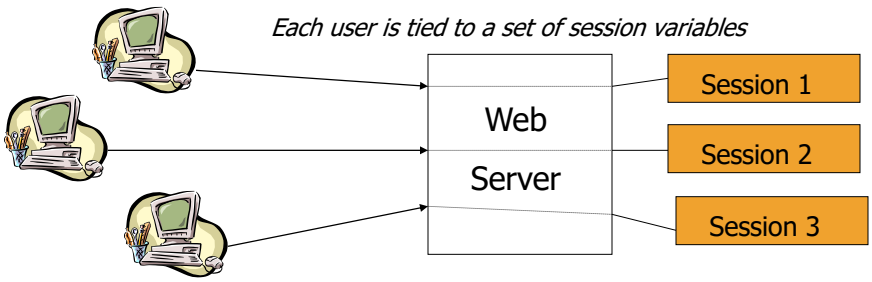
# Session State

- Store client state on Web server
- Client cookie indexes a particular *session store*

- ASP.NET: `Session[key] = value;`

25

# Session Variables




26

## Session Variables

- Performance / Memory Consumption
  - Resources on Web server
- Alternatives:
  - Use backend database
    - in database cluster
  - Use separate session server
    - in Web server cluster

27

## Data Store

- Session variables: use Web server memory (default) for session store
- Amazon: use separate database to store shopping cart
  - “Permanent”
  - ASP.NET: Profile subsystem

28

## Identifying Session State

- Cookie
  - Anti-pattern for REST
- Query String parameter
  - Cookie-less server

```
http://localhost/Controller/Action?
      Name1=Value1&Name2=Value2&...&NameN=ValueN
```

  - Look up query string values:

```
string qsValue = Request.QueryString["name"];
```

29

## Identifying Session State

- REST
  - Identify shared session state as a resource
  - Every resource has a URI

```
http://domain/Shop/Checkout/session-id

public ActionResult Checkout(int id) {
    ...
}
```

30

# DATA ATTRIBUTES AND VALIDATION

31

## Validation

- Model:

```
public class Image {
   [Required(ErrorMessage="A caption is required")]
   public String Caption { get; set; };
}
```

- View:

```
<input asp-for="Caption"/>
<span asp-validation-for="Caption"/>
```

- HTML Output:

```
<span class="field-validation-error"
      data-valmsg-for="Caption"
      data-valmsg-replace="true">
   A caption is required</span>
```

32

## Data Attributes

- Include namespace:

```
using System.ComponentInfo.DataAnnotations;
```

- Specify data validation in the model
- `ModelState` captures state of validation

```
ModelState.IsValid()
```

33

## Image Model

```
public class Image {
   public String Id { get; set; };
   public String Caption { get; set; };
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

34

# Required

```
public class Image {
   [Required]
   public String Id { get; set; };
   [Required]
   public String Caption { get; set; };
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

35

# StringLength

```
public class Image {
   [Required]
   public String Id { get; set; };
   [Required]
   [StringLength(40)]
   public String Caption { get; set; };
   [StringLength(200)]
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

36

# RegularExpression

```
public class Image {
   [Required]
   [RegularExpression(@”[a-zA-Z0-9_]+”)]
   public String Id { get; set; };
   [Required]
   [StringLength(40)]
   public String Caption { get; set; };
   [StringLength(200)]
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

37

# ErrorMessage

```
public class Image {
   [Required]
   [RegularExpression(@”[a-zA-Z0-9_]+”,
          ErrorMessage=“Not a valid identifier”]
   public String Id { get; set; };
   [Required(
          ErrorMessage=“Please provide a caption.”]
   [StringLength(40,
          ErrorMessage=“Caption is too long.”)]
   public String Caption { get; set; };
   [StringLength(200)]
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

38

# Display

```
public class Image {
   [Required]
   [RegularExpression(@”[a-zA-Z0-9_]+”)]
   [Display(Name=“Image identifier”)]
   public String Id { get; set; };
   [Required(ErrorMessage=“...”)]
   [StringLength(40, ErrorMessage=“...”)]
   public String Caption { get; set; };
   [StringLength(200)]
   public String Description { get; set; };
   [Display(Name=“Date photo taken”)]
   public Date DateTaken { get; set; };
   public String UserId { get; set; };
}
```

39

# ScaffoldColumn

```
public class Image {
   [Required]
   [RegularExpression(@”[a-zA-Z0-9_]+”)]
   [Display(Name=“Image identifier”)]
   public String Id { get; set; };
   [Required(ErrorMessage=“...”)]
   [StringLength(40, ErrorMessage=“...”)]
   public String Caption { get; set; };
   [StringLength(200)]
   public String Description { get; set; };
   [Display(Name=“Date photo taken”)]
   public Date DateTaken { get; set; };
   [ScaffoldColumn(false)]
   public String UserId { get; set; };
}
```

40

# DataType

Datatypes:
- Password
- Currency
- Date
- Time
- MultilineText

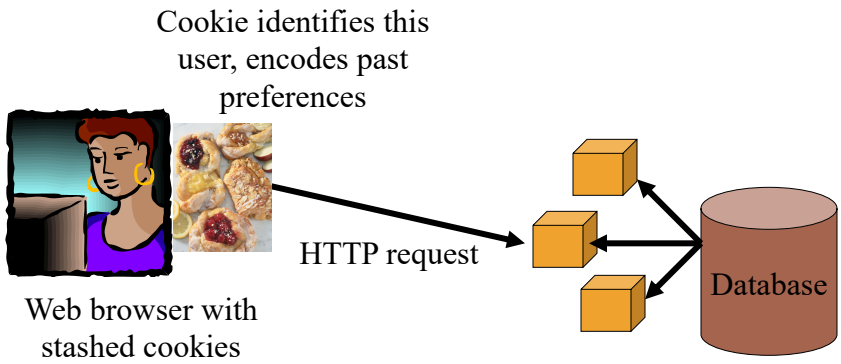
```
public class Image {
   [Required]
   [RegularExpression(@"[a-zA-Z0-9_]+
   [Display(Name="Image identifier")]
   public String Id { get; set; };
   [Required(ErrorMessage="...")]
   [StringLength(40, ErrorMessage="...")]
   public String Caption { get; set; };
   [StringLength(200)]
   public String Description { get; set; };
   [Display(Name="Date photo taken")]
   [DataType(DataType.Date)]
   public Date DateTaken { get; set; };
   [ScaffoldColumn(false)]
   public String UserId { get; set; };
}
```

41

# DATABASES: MOTIVATION

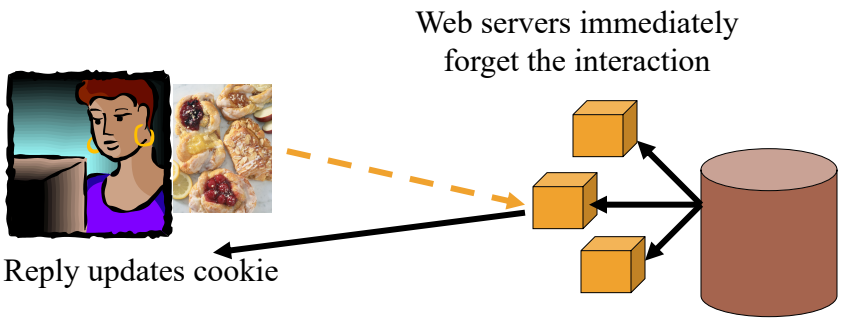
42

# State on the Web




43

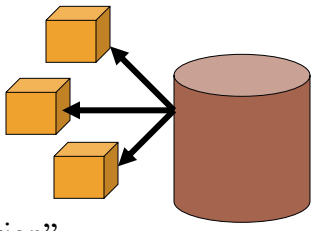
# State on the Web




44

# State on the Web

Web servers have no memory of the interaction



Purchase is a "transaction" on the database



# Motivation for database systems

- Example:
  - File-based data processing
  - Data stored in files
  - Files are composed of records
  - Indexes speed up record retrieval

```
select EmployeeFile
    assign to "EMPLOY.DAT"
    organization is indexed
    access mode is random
    record key is EmployeeID
select DeptFile
    assign to "DEPTS.DAT"
    organization is indexed
    access mode is random
    record key is DeptID.

FD EmployeeFile.
01 EmployeeRecord.
    02 EmployeeID pic 9(10).
    02 EmployeeName pic X(20).
    02 Dept       pic 9(5).
FD DeptFile.
01 DeptRecord.
    02 DeptID     pic 9(5).
    02 DeptHead   pic 9(10).
```

# Motivation for database systems

- What's missing?
  - Integrity constraints
    - Need a data model!
  - Ad-hoc queries & analysis
    - Need a query language!
    - Efficient execution!
  - Concurrency control
  - Failure recovery
  - Access control

```
select EmployeeFile
   assign to "EMPLOY.DAT"
   organization is indexed
   access mode is random
   record key is EmployeeID
select DeptFile
   assign to "DEPTS.DAT"
   organization is indexed
   access mode is random
   record key is DeptID.

FD EmployeeFile.
01 EmployeeRecord.
   02 EmployeeID pic 9(10).
   02 EmployeeName pic X(20).
   02 Dept       pic 9(5).
FD DeptFile.
01 DeptRecord.
   02 DeptID     pic 9(5).
   02 DeptHead   pic 9(10).
```

47

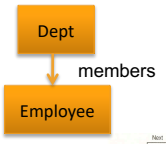
# Data Models

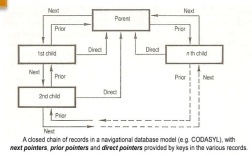
- Tools for describing:
  - Data
  - Relationships
  - Constraints
- Schema: logical description
  - Entities, e.g. departments and employees
  - Relationships between them
- Instance: actual contents of database

48

# Data Models

- Hierarchical
  - Database is a “tree”
  - Ex: IMS, Windows Registry
- CODASYL
  - Database is a “network”
- Relational
  - Database is a set of tables
  - Ex: DB2, Oracle, MySQL, …
- XML/JSON
  - Database is a document
  - “Semi-structured data”



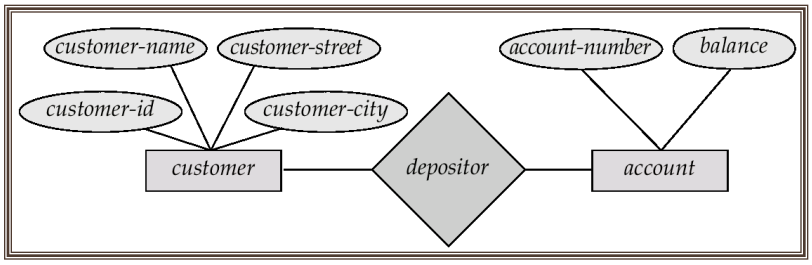




A closed chain of records in a navigational database model (e.g. CODASYL), with *next pointers*, *prior pointers* and *direct pointers* provided by keys in the various records.

| EmpNo | Name | Dept |
|---|---|---|
| 123456 | J Smith | Marketing |
| 654321 | S Bloggs | HR |
| 894533 | M Ploog | Finance |



# Entity-Relationship Data Model



# Relational Data Model

| *Customer-id* | *customer-name* | *customer-street* | *customer-city* | *account-number* |
|---|---|---|---|---|
| 192-83-7465 | Johnson | Alma | Palo Alto | A-101 |
| 019-28-3746 | Smith | North | Rye | A-215 |
| 192-83-7465 | Johnson | Alma | Palo Alto | A-201 |
| 321-12-3123 | Jones | Main | Harrison | A-217 |
| 019-28-3746 | Smith | North | Rye | A-201 |

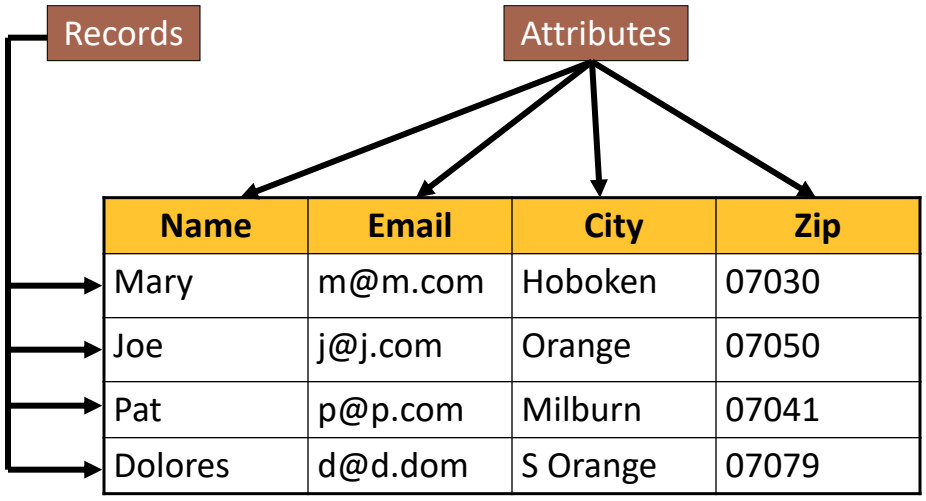


# RELATIONAL DATA MODEL

# Relational Data Model

- A database is comprised of tables
  - Ex: Customers and Products tables
- A table is comprised of one of more columns
  - Attributes
  - Ex: Customers: Name, Address, City, State, etc.
  - Each column has an associated data type
- Each table has zero or more records

# Relational Data Model




| Name | Email | City | Zip |
|---|---|---|---|
| Mary | m@m.com | Hoboken | 07030 |
| Joe | j@j.com | Orange | 07050 |
| Pat | p@p.com | Milburn | 07041 |
| Dolores | d@d.dom | S Orange | 07079 |

## Primary Keys

- A column that uniquely identifies each record in a table
  - Ex: customer ID, product ID

55

## Identity Columns

- If no natural primary key column
- Create a numeric column
  - Mark as primary key
  - Mark as identity column
- Values generated by DBMS

56

# Identity Columns

| Cust ID | Name | Email | City | Zip |
|---|---|---|---|---|
| 1 | Mary | m@m.com | Hoboken | 07030 |
| 2 | Joe | j@j.com | Orange | 07050 |
| 3 | Pat | p@p.com | Milburn | 07041 |
| 4 | Dolores | d@d.dom | S Orange | 07079 |

57

# RELATIONSHIPS IN THE RELATIONAL MODEL

58

## Rela

**Data Duplication**
- Difficult to maintain
- Wasted space in storage
- Hard to query efficiently

| EmpID | Name | Email | Dept | Manager |
|---|---|---|---|---|
| 1 | Mary | m@m.com | IT | Mary |
| 2 | Joe | j@j.com | IT | Mary |
| 3 | Pat | p@p.com | Sales | Pat |
| 4 | Dolores | d@d.dom | Executive | Dolores |
| 5 | Tina | t@t.com | HR | Tina |
| 6 | Bruce | b@b.com | Executive | Dolores |

## Relationships

| EmpID | Name | Email | DeptID |
|---|---|---|---|
| 1 | Mary | … | 1 |
| 2 | Joe | … | 1 |
| 3 | Pat | … | 2 |
| 4 | Dolores | … | 3 |
| 5 | Tina | … | 4 |
| 6 | Bruce | … | 3 |

| DeptID | Name |
|---|---|
| 1 | IT |
| 2 | Sales |
| 3 | Exec |
| 4 | HR |

# Relationships

- Three kinds of relationships:
  - One-to-one
    - Ex: Customer preferences

61

# One-to-one

| CustID | Name | Email |
|---|---|---|
| 1 | Mary | … |
| 2 | Joe | … |
| 3 | Pat | … |
| 4 | Dolores | … |
| 5 | Tina | … |
| 6 | Bruce | … |

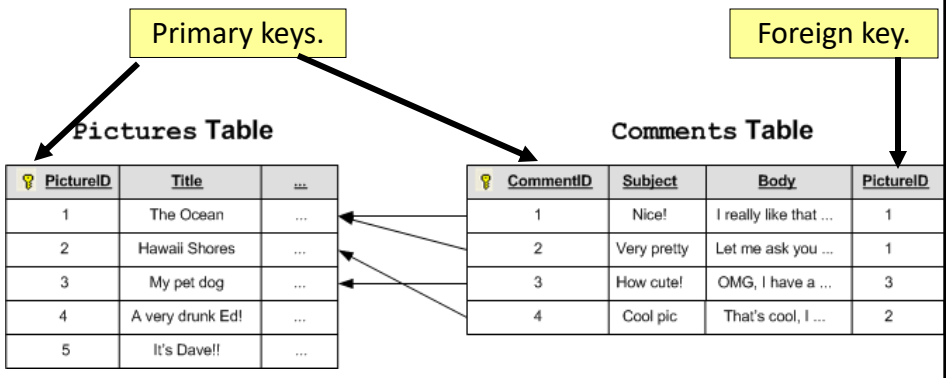
| CustID | SrDisp | ShOpt |
|---|---|---|
| 1 | 10 | USPS |
| 2 | 10 | UPS |
| 3 | 25 | USPS |
| 4 | 100 | FedEx |
| 6 | 25 | USPS |

# Relationships

- Three kinds of relationships:
  - One-to-one
    - Ex: Customer preferences
  - One-to-many
    - Ex: Customer posts on a blog
    - Ex: Replies to a blog post
    - Ex: Customer comments on a picture gallery

63

# One-to-many



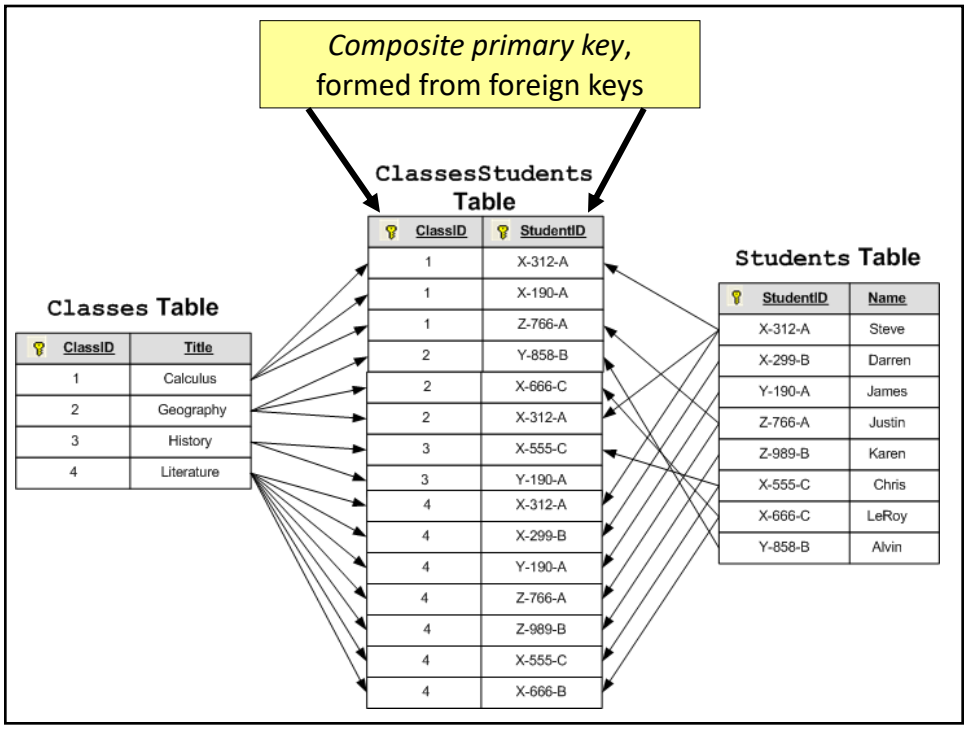

| PictureID | Title | ... |
|---|---|---|
| 1 | The Ocean | ... |
| 2 | Hawaii Shores | ... |
| 3 | My pet dog | ... |
| 4 | A very drunk Ed! | ... |
| 5 | It's Dave!! | ... |

| CommentID | Subject | Body | PictureID |
|---|---|---|---|
| 1 | Nice! | I really like that ... | 1 |
| 2 | Very pretty | Let me ask you ... | 1 |
| 3 | How cute! | OMG, I have a ... | 3 |
| 4 | Cool pic | That's cool, I ... | 2 |

# Relationships

- Three kinds of relationships:
  - One-to-one
    - Ex: Customer preferences
  - One-to-many
    - Ex: Customer posts on a blog
    - Ex: Replies to a blog post
    - Ex: Customer comments on a picture gallery
  - Many-to-many
    - Ex: Students enrolled in courses

65



# Referential Integrity

- Don't allow "orphan records"
  - Ex: Comment for non-existent picture
  - Foreign key constraint
    - On foreign key
    - On record deletion

**Pictures Table**

| PictureID | Title | ... |
|---|---|---|
| 1 | The Ocean | ... |
| 2 | Hawaii Shores | ... |
| 3 | My pet dog | ... |
| 4 | A very drunk Ed! | ... |
| 5 | It's Dave!! | ... |

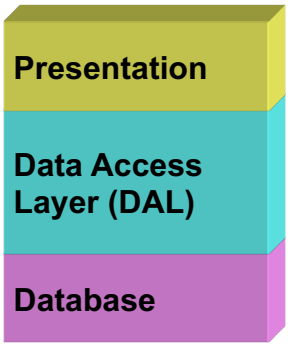
**Comments Table**

| CommentID | Subject | Body | PictureID |
|---|---|---|---|
| 1 | Nice! | I really like that ... | 1 |
| 2 | Very pretty | Let me ask you ... | 1 |
| 3 | How cute! | OMG, I have a ... | 3 |
| 4 | Cool pic | That's cool, I ... | 2 |

# APPLICATION ARCHITECTURE

# Application Architecture

- Presentation Layer (Web pages)
- Data Access Layer (DAL)
- Database

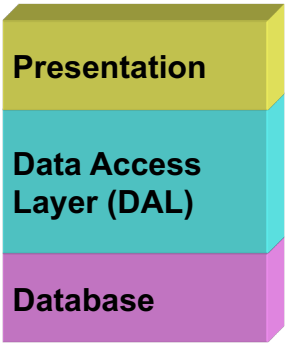



69

# Data Access Layer

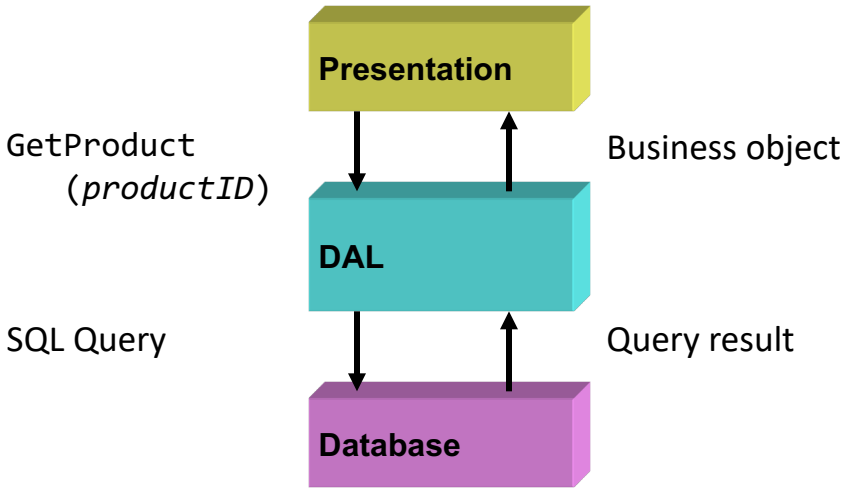
- Data Access Layer API

```
GetProduct
    (productID)

CalculateShipping
    (shoppingCartID)

PlaceOrder
    (orderID)
```




70

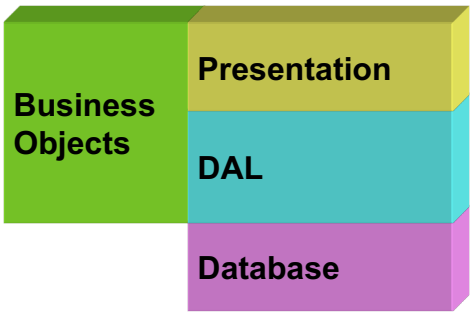
# DAL in Architecture




71

# Business Objects

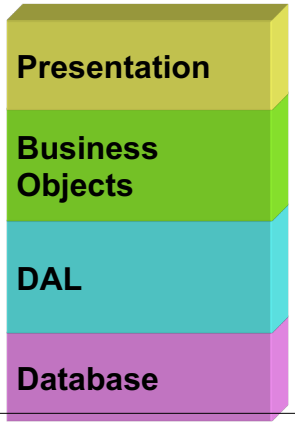
- May be orthogonal to the other layers
- Domain entities
- Domain operations implemented directly in the DAL
- *Stored procedures*




Example code:
```
float shipping = DAL.CalculateShipping(cartID);
```

# Business Objects

- May be a layer in the architecture
- Domain operations encapsulated in the objects
- *Domain-driven design (cf CS548)*




Example code:

```
ShoppingCart myCart = DAL.getShoppingCart(cartID);
float shipping = myCart.calculateShipping();
```

# IMPLEMENTING THE DAL

74

## Data Access Design Patterns

- Plain Old CLR Object (POCO)
  - Business logic only
- Repository
  - CRUD interface
  - Obtained via dependency injection (DI)
- Object-Relational Mapping (ORM)
  - Entity Framework

75

## Object Relational Impedance Mismatch

- Granularity
  - Common data model, different behaviors
- Inheritance
  - Subclasses vs flat tables
- Identity
  - Object identity vs primary key
- Associations
  - Directionality: References vs foreign keys
- Data navigation
  - Walk object graph vs explicit queries

76

## Data Access Approach

- Database First
  - Configuration from schema
  - Classes inherit from `EntityObject`
- Model First
  - Schema from model
  - Classes inherit from `EntityObject`
- Code First
  - Persistence ignorance

77

## ENTITY FRAMEWORK CODE FIRST

78

## EF Code First

- Convention over Configuration
- Table name based on class name
  - class `Product` ⇒ table `Products`
- Column names from property names
- Primary keys based on properties
  - `ID` or `classNameID`
- Default connection string
  - Name of `DataContext` class

79

## EF Code First

- Annotate POCOs
  - `Table, Column`
  - `ConcurrencyCheck`
  - `DatabaseGenerated`
  - `Key, ForeignKey`
  - `InverseProperty`
  - `Required`
  - `MaxLength, MinLength, StringLength`
  - `Timestamp`

80

# Image Model

- Model for photographic image

```
public class Image {
   public String Id { get; set; };
   public String Tag { get; set; };
   public String Caption { get; set; };
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public String User { get; set; };
}
```

81

# Image Model

- Model for photographic image

Navigational property

Foreign key property

```
public class Image {
   public int Id { get; set; };
   public Tag Tag { get; set; };
   public int TagId { get; set; };
   public String Caption { get; set; };
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public User User { get; set; };
   public int UserId { get; set; };
}
```

82

# Image Model

```
public class Image {
   [Key]
   public int Id { get; set; };
   public Tag Tag { get; set; };
   [ForeignKey("Tag")]
   public int TagId { get; set; };
   public String Caption { get; set; };
   public String Description { get; set; };
   public Date DateTaken { get; set; };
   public User User { get; set; };
   [ForeignKey("User")]
   public int UserId { get; set; };
}
```

83

# Image Model

```
public class Image {
   [Key]
   public virtual int Id { get; set; };
   public virtual Tag Tag { get; set; };
   [ForeignKey("Tag")]
   public virtual int TagId { get; set; };
   public virtual String Caption { get; set; };
   public virtual String Description
                        { get; set; };
   public virtual Date DateTaken { get; set; };
   public virtual User User { get; set; };
   [ForeignKey("User")]
   public virtual int UserId { get; set; };
}
```

84

# Tag Model

```
public class Tag {
   [Key]
   public virtual int Id { get; set; };
   public virtual String Name { get; set; };
}
```

85

# Tag Model

```
public class Tag {
   [Key]
   public virtual int Id { get; set; };
   public virtual String Name { get; set; };
   public virtual List<Image> Images { get; set; };
}
```

86

# User Model

```
public class User {
   [Key]
   public virtual int Id { get; set; };
   public virtual String Userid { get; set; };
   public virtual String Password { get; set; };
   public virtual String Name { get; set; };
}
```

87

# User Model

```
public class User {
   [Key]
   public virtual int Id { get; set; };
   public virtual String Userid { get; set; };
   public virtual String Password { get; set; };
   public virtual String Name { get; set; };
   public virtual List<Image> Images { get; set; }
}
```

88

## ACCESSING THE DATABASE

89

## Adding EF Migrations

```
<Project Sdk="Microsoft.NET.Sdk.Web">
<PropertyGroup>
<TargetFramework>netcoreapp2.0</TargetFramework>
</PropertyGroup>
<ItemGroup> <Folder Include="wwwroot\" /> </ItemGroup>
<ItemGroup>
  <PackageReference
     Include="Microsoft.AspNetCore.All"
     Version="2.0.0" />
  <DotNetCliToolReference
     Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
     Version="2.0.0" />
</ItemGroup>
</Project>
```

90

# Adding EF Migrations

- From Developer PowerShell

```
dotnet ef migrations add Initial
```

```
PS Microsoft.PowerShell.Core\FileSystem::\\Mac\Home\Documents\Teach\CS526\Classes\2019-Fall\Assignmen
ts\Assignment2-Code\ImageSharingWithModel\ImageSharingWithModel> dotnet ef migrations add Initial
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.1.11-servicing-32099 initialized 'ApplicationDbContext' using provider
'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Done. To undo this action, use 'ef migrations remove'
```

91

# Database Context

- Database connection session:

```
using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext {
   public DbSet<Image> Images { get; set; }
   public DbSet<User> Users { get; set; }
   public DbSet<Tag> Tags { get; set; }
}
```

92

## Lazy vs Eager Loading

- Lazy loading (N+1 problem):

```
ApplicationDbContext db;
var images = db.Images;
```

- Eager loading:

```
ApplicationDbContext db;
var images = db.Images.Include(i ⇒ i.User)
                      .Include(i ⇒ i.Tag);
```

93

## Querying the Database

- LINQ query

```
ApplicationDbContext db;
var allImages = from image in db.Images
                orderBy image.DateTaken
                select image;
return View(allImages.ToList());
```

94

## View for Query Result (1/2)

```
@model IEnumerable<ImageSharing.Models.Image>

<p> <a asp-action="Create">Create New</a> </p>

<table>
   <tr>
      <th>Caption</th>
      <th>Tag</th>
      <th>Uploader</th>
      <th>Links</th>
   </tr>
   @foreach (var item in Model) {
      <tr> ... </tr>
   }
</table>
```

95

## View for Query Result (2/2)

```
@foreach (var item in Model) {
<tr>
  <td>@item.Caption</td>
  <td>@item.Tag.Name</td>
  <td>@item.User.Username</td>
  <td>
    <a asp-action="Edit"
       asp-route-id="Image.Id">Edit</a>
    <a asp-action="Details"
       asp-route-id="Image.Id">Details</a>
    <a asp-action="Delete"
       asp-route-id="Image.Id">Delete</a>
 </td>
</tr>
}
```

96

# Database Connection

- Connection String (appsettings.json)

```
{
  ...,
  "Data": {
    "ImageSharingDB": {
      "ConnectionString" :
          "Server=(localdb)\\MSSQLLocalDB;
           Database=ImageSharingDB;
           Trusted_Connection=True;
           MultipleActiveResultSets=true"
    }
  }
}
```

97

# Dependency Injection

- Specify DB context as a service (Startup.cs)

```
public void ConfigureServices(IServiceCollection services) {
   ...
   services.AddDbContext<ApplicationDbContext>
     (options => options.UseSqlServer
         (Configuration["Data:ImageSharingDB:ConnectionString"]));
}
```

- Inject into controller

```
public void ImagesController : Controller {
  protected readonly ApplicationDbContext db;
  public ImagesController(ApplicationDbContext db) {
    this.db = db;
  }
  ...
}
```

98

## Seeding the Database (1/2)

```
public static class ApplicationDbInitializer {

  public static void Seed(ApplicationBuilder app) {
     ApplicationDbContext db =
        app.ApplicationServices
           .GetRequiredService<ApplicationDbContext>();

     db.Database.Migrate();

     db.Users.Add(new User {Username="johndoe",...});
     db.Tags.Add(new Tag {Name="architecture"});
  }

}
```

99

## Seeding the Database (2/2)

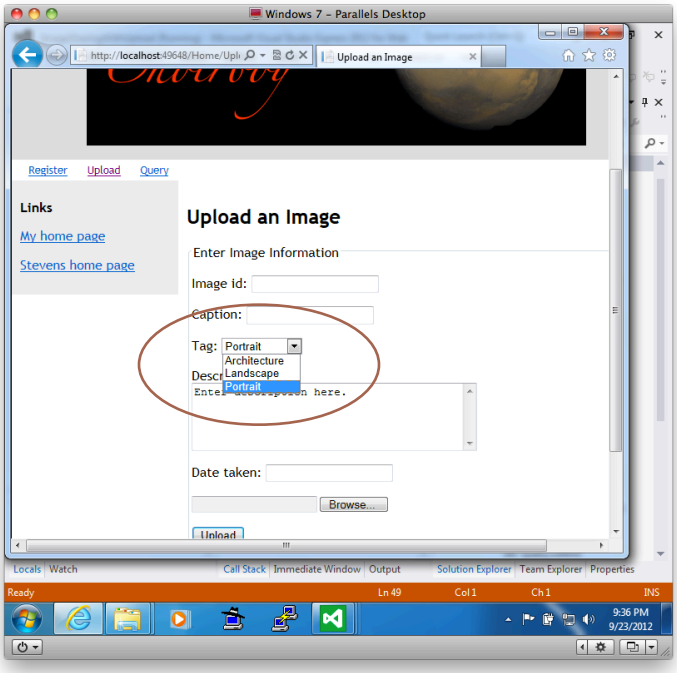
- Initialize in startup (Startup.cs):

```
public void Configure(IApplicationBuilder app,
                      IHostingEnvironment env) {
   ...
   app.UseMvc(routes => ...);
   ApplicationDbInitializer.Seed(app);
}
```

100

# EDIT ACTION

101




102

# Model for Tags

| Id | Name |
|---|---|
| 1 | Architecture |
| 2 | Landscape |
| 3 | Portrait |

- Tag for a photographic image

```
public class Tag {
   public int Id { get; set; };
   public String Name { get; set; };
}
```

- Database of tags

```
IEnumerable<Tag> tags;
```

103

# ListItem

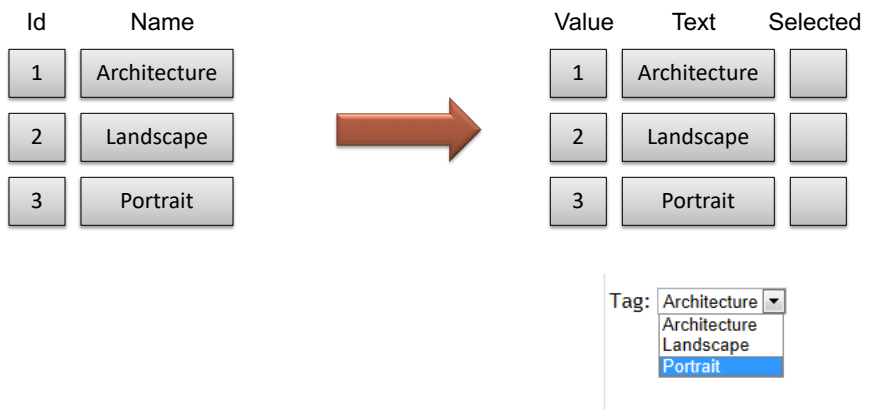
| Value | Text | Selected |
|---|---|---|
| 1 | Architecture | |
| 2 | Landscape | |
| 3 | Portrait | |

- Item in a dropdown list, radio button list, checkbox list, etc


```
public class ListItem {
   public string Value { get; set; };
   public string Text { get; set; };
   public boolean Selected { get; set; }
}
```

104

# SelectList Helper




# DropDownList and ListBox

- Building an Edit Form

```
public ActionResult Edit(int imageId) {
    Image image = ...;
    ViewBag.Tags =
        new SelectList( tags,
                        "Id", "Name",
                        image.TagId );
    return View(image);
}
```

IEnumerable<Tag>

Property name for selected value

Property name for displayed value

Initial selection

106

# Edit Action

| Id | Name | Selected |
|---|---|---|
| 1 | Architecture | |
| 2 | Landscape | |
| 3 | Portrait | |

- Building an Edit Form

```
public ActionResult Edit(int id) {
  Image image = db.Images.Find(id);
  ViewBag.Tags =
    new SelectList(db.Tags,
              "Id", "Name", image.TagId);
  return View(image);
}
```

107

# Edit View

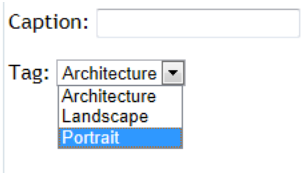
Caption:

Tag: Architecture
Architecture
Landscape
Portrait

```
<form method="post" ...>
  <fieldset>
    <legend> Edit Image</legend>
    <p> <label asp-for="TagId"/>
        <select asp-for="TagId"
            asp-items="ViewBag.Tags as SelectList"/>
    </p>

    <p> <label asp-for="Caption"/>
        <input asp–for="Caption"/>
        <span asp-validation-for="Caption"/>
    </p>

    <input type="submit" value="Save" />
  </fieldset>
</form>
```

108

# Edit View




```
<form method="post" ...>
  <fieldset>
    <legend> Edit Image</legend>
    <p> <label asp-for="TagId"/>
        <select asp-for="TagId"
            asp-items="ViewBag.Tags as SelectList"/>
    </p>

    <p> <label asp-for="Caption"/>
        <input asp-for="Caption"/>
        <span asp-validation-for="Caption"/>
    </p>

    <input type="submit" value="Save" />
  </fieldset>
</form>
```

109

# Edit View

| Value | Text | Selected |
|---|---|---|
| 1 | Architecture | |
| 2 | Landscape | |
| 3 | Portrait | |

```
<form method="post" ...>
  <fieldset>
    <legend> Edit Image</legend>
    <p> <label asp-for="TagId"/>
        <select asp-for="TagId"
            asp-items="ViewBag.Tags as SelectList"/>
    </p>

    <p> <label asp-for="Caption"/>
        <input asp-for="Caption"/>
        <span asp-validation-for="Caption"/>
    </p>

    <input type="submit" value="Save" />
  </fieldset>
</form>
```

110

# Edit Action

- Building an Edit Form

```
public ActionResult Edit(int id) {
  Image image = db.Images.Find(id);
  ViewBag.Tags =
    new SelectList(db.Tags,
             "Id", "Name", image.TagId);
  return View(image);
}
```

111

# Edit Action

```
public class ImageEditModel {
  public Image ImageToEdit { get; set; }
  public SelectList Tags { get; set; }
}

public ActionResult Edit(int id) {
  ImageEditModel em = new ImageEditModel();
  em.ImageToEdit = db.Images.Find(id);
  em.Tags =
    new SelectList(db.Tags,
             "Id", "Name", image.TagId);
  return View(em);
}
```

112

# Edit View

```
<form method="post" ...>
  <fieldset>
    <legend>Edit Image</legend>

    <p> <label asp-for="ImageToEdit.TagId">Tag</label>
        <select asp-for="ImageToEdit.TagId"
                asp-items="Tags" /> ...
    </p>
    <input type="submit" value="Save" />
  </fieldset>
</form>
```

113

# Edit Form

```
<form action="/Image/Edit/17" method="post">
  <label for="TagId">Tag</label>
  <select id="TagId" name="TagId">
    <option value=""></option>
    <option value="1" selected="selected">
      Architecture</option>
    <option value="2">Landscape</option>
  </select>

  <input class="text-box single-line"
    id="Caption" name="Caption"
    type="text" value="..." />

  <p><input type="submit" value="Save" /></p>
</form>
```

114

# Edit Processor

```
[HttpPost]
Public ActionResult Edit(Image image) {

   if (ModelState.IsValid) {
      db.Entry(image).State = EntityState.Modified;
      db.SaveChanges();
      return RedirectToAction("Index");
   }

   ImageEditModel em = new ImageEditModel();
   em.Tags = new SelectList(...);
   em.ImageToEdit = image;
   return View(em);
}
```

115

# Edit Processor

```
[HttpPost]
Public ActionResult Edit(int id,
                         FormCollection collection) {
   Image image = ImageSharingDb.Images.Find(id);
   TryUpdateModelAsync(image);
   if (ModelState.IsValid) {
      db.Entry(image).State = EntityState.Modified;
      db.SaveChanges();
      return RedirectToAction("Index");
   }

   ImageEditModel em = new ImageEditModel();
   em.Tags = ...;  em.ImageToEdit = ...;
   return View(em);
}
```