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

```
public ActionResult Upload(Image image) {

   JavaScriptSerializer serializer =
      new JavaScriptSerializer();
   String jsonData = serializer.Serialize(image);

   String fileName =
      Server.MapPath (“~/App_Data/Image_Info/”
                      + image.Id + “.js”);
   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) {
      RedirectToAction ("Error", "Home",
            new { userid=userid, errid="upload" });
   }
}
```

Retrieving Data

```
Public ActionResult Query(string id) {
   ...
   String jsonData = File.ReadAllText(filename);
   JavaScriptSerializer serializer =
      new JavaScriptSerializer();

   Image image=
      serializer.Deserialize<Image>(jsonData);

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

Retrieving Data

```
Public ActionResult Query(string id) {
   ...
   if (File.Exists(filename)) {
      String jsonData = File.ReadAllText(filename);
      JavaScriptSerializer serializer =
         new JavaScriptSerializer();

      Image image =
         serializer.Deserialize<Image>(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();
}
```

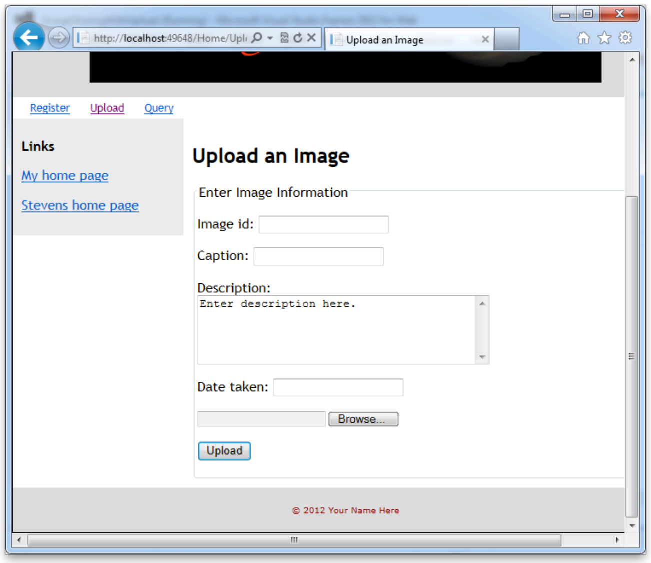
8

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



Upload an Image

Enter Image Information

Image id:

Caption:

Description:

Enter description here.

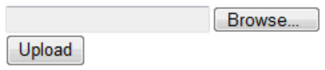
Date taken:

10

File Upload Form

- Generate the form:

```
@{ Html.BeginForm(“Upload”, “Home”,
        FormMethod.Post,
        new {enctype="multipart/form-data"}) }
    <input type=“file” name=“ImageFile” />
    <br/>
    <input type=“submit” value=“Upload”
           name=“Submit” id=“Submit” />
@{ Html.EndForm() }
```

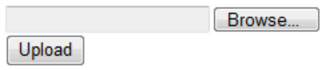


11

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

Managing File on Server

- Controller class
- Request
- Request.Files
- Request.Files[filename] (type HttpPostedFileBase)

Processing Upload

```
[HttpPost]
public ActionResult Upload (Image image) {
   HttpPostedFileBase file = Request.Files[0];
   if (file != null && file.ContentLength > 0) {
       file.SaveAs(
          Server.MapPath("~/Content/Images/"
                               + image.id + ".jpg"));
   }
   return View();
}
```

Processing Upload

```
[HttpPost]
public ActionResult Upload
      (Image image, HttpPostedFileBase ImageFile) {
   if (ImageFile != null &&
       ImageFile.ContentLength > 0) {
       ImageFile.SaveAs(
         Server.MapPath("~/Content/Images/"
                         + image.id + ".jpg"));
   }
   return View();
}
```

15

Processing Upload

```
[HttpPost]
public ActionResult Upload (Image image,
                            HttpPostedFileBase ImageFile) {
   if (ImageFile != null && ImageFile.ContentLength > 0) {
      if (ImageFile.ContentLength > LIMIT) {
         // Report an error on file size
      } else if (ImageFile.ContentType != "image/jpeg") {
         // Error on content type; but can be faked!
      } else {
         ImageFile.SaveAs(
            Server.MapPath("~/Content/Images/"
                           + image.id + ".jpg"));
      }
   }
   return View();
}
```

16

Validating an Image File

- Turn the file into an image

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

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

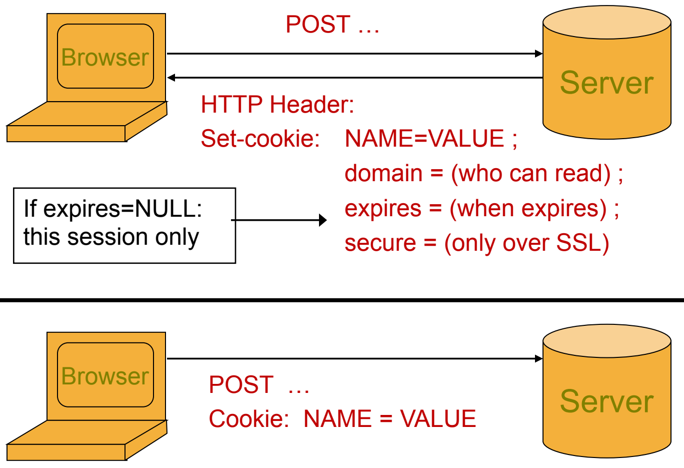
Session State

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

20

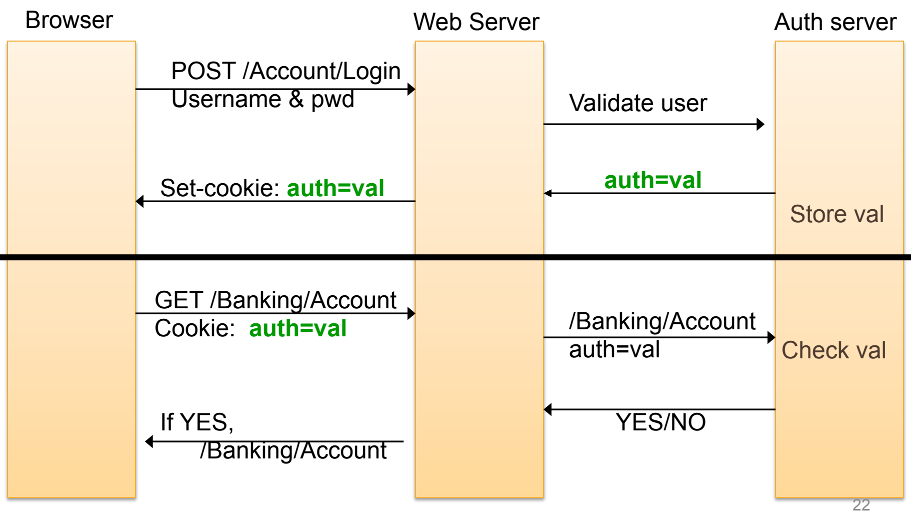
Cookies

- Used to store state on user's machine



21

Cookie authentication



22

Cookies

- Writing a value to a cookie:

```
HttpCookie cookie =
  new HttpCookie(“cookieName”);
cookie.Expires =
  DateTime.Now.AddMonths(3);
Response.Cookies.add (cookie);
```

- Reading the value from a cookie:

```
HttpCookie cookie =
  Request.Cookies.Get(“cookieName”);
```

23

Cookies

- Cookies can hold several values:

```
HttpCookie cookie =
        new HttpCookie(“myCookie”);
cookie[“UserId”] = UserId;
cookie[“ADA”] = isADA ? “true” : “false”;
```

24

Cookies

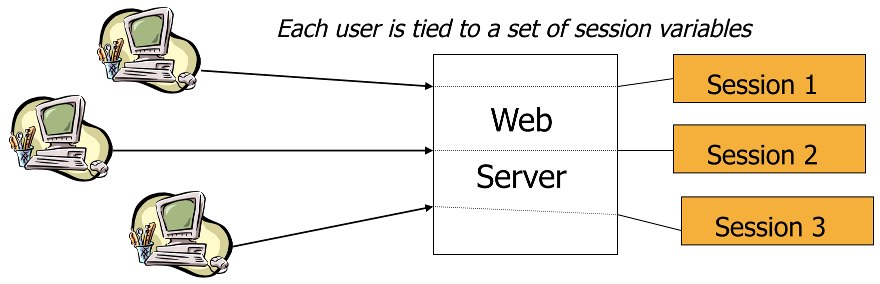
- Cookies are sent back and forth in plain-text.
- Cookies are stored on the client’s computer.
- Never store sensitive information in cookies.

Session State

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

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

Session Variables



Session Variables

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

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

Identifying Session State

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

    `http://localhost/Controller/Action?`
    $\texttt{Name}_1\texttt{=Value}_1\texttt{\&Name}_2\texttt{=Value}_2\texttt{\&...\&Name}_N\texttt{=Value}_N$
  - Look up query string values:

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

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) {
   ...
}
```

31

DATA ATTRIBUTES AND VALIDATION

32

Validation

- Controller:

```
public ActionResult Edit(int imageId) {
   Image image = ...;
   return View(image);
}
```

- View:

```
@Html.ValidationMessage(“Caption”,
                        “A caption is required!”);
```

- HTML Output:

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

33

Validation

- Model:

```
public class Image {
   [Required]
   public String Caption { get; set; };
}
```

- View:

```
@Html.ValidationMessage(“Caption”,
                        “A caption is required!”);
```

- HTML Output:

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

34

Data Attributes

- Include namespace:
  `using System.ComponentInfo.DataAnnotations;`
- Specify data validation in the model
- `ModelState` captures state of validation
  `ModelState.IsValid()`

35

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; };
}
```

36

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; };
}
```

37

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; };
}
```

38

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; };
}
```

39

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; };
}
```

40

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; };
}
```

41

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; };
}
```

42

DataType

```
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; };
}
```

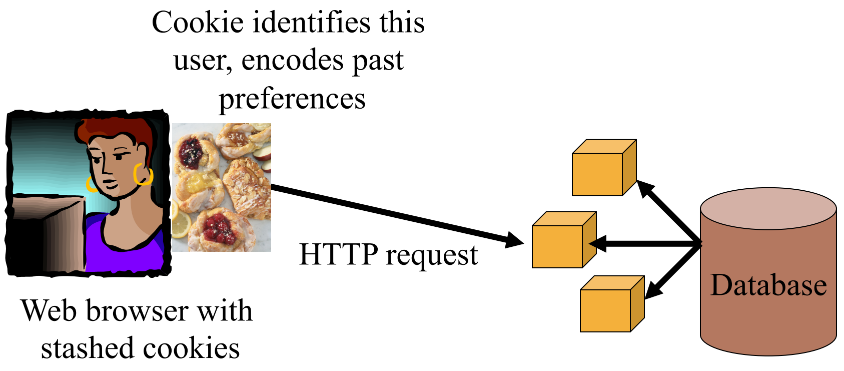
Datatypes:
- Password
- Currency
- Date
- Time
- MultilineText

43

DATABASES: MOTIVATION

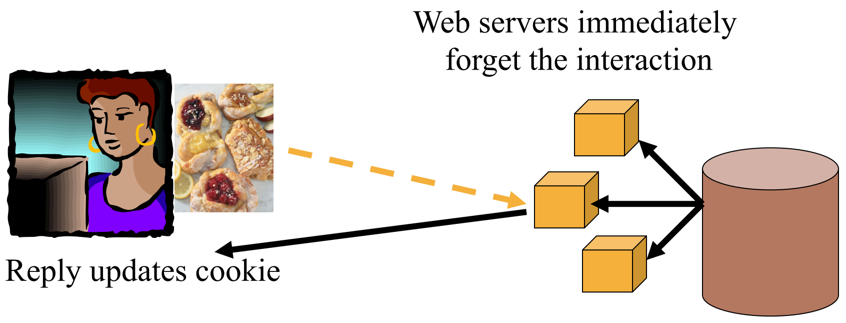
44

State on the Web


Cookie identifies this user, encodes past preferences
HTTP request
Database
Web browser with stashed cookies

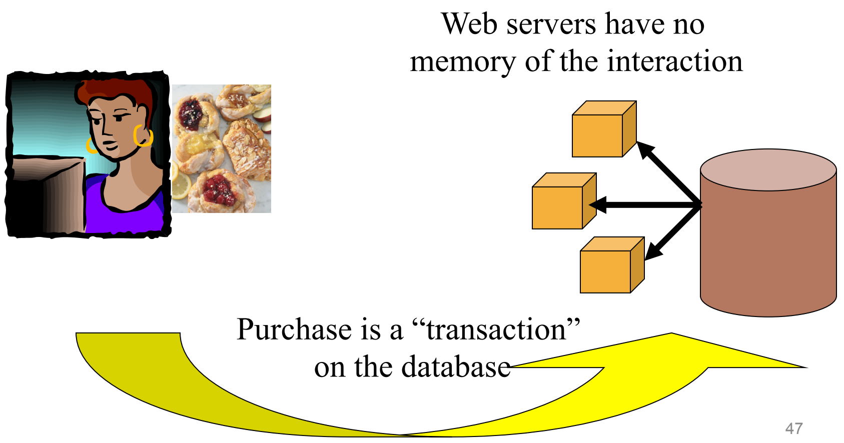
45

State on the Web



46

State on the Web


Web servers have no memory of the interaction
Purchase is a "transaction" on the database

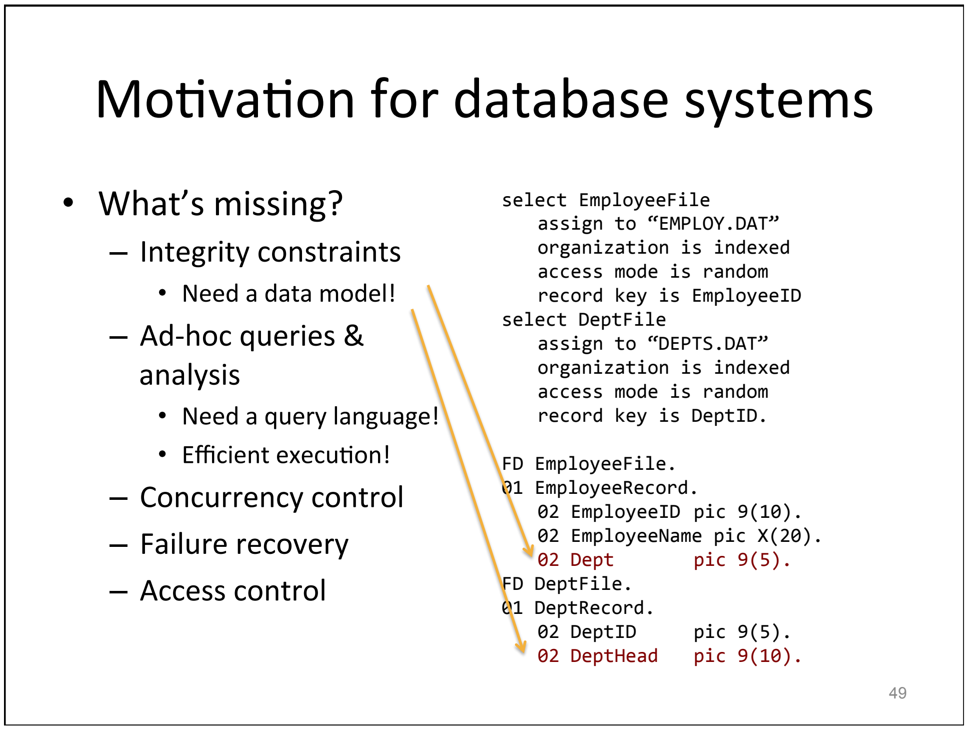
47

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).
```

48

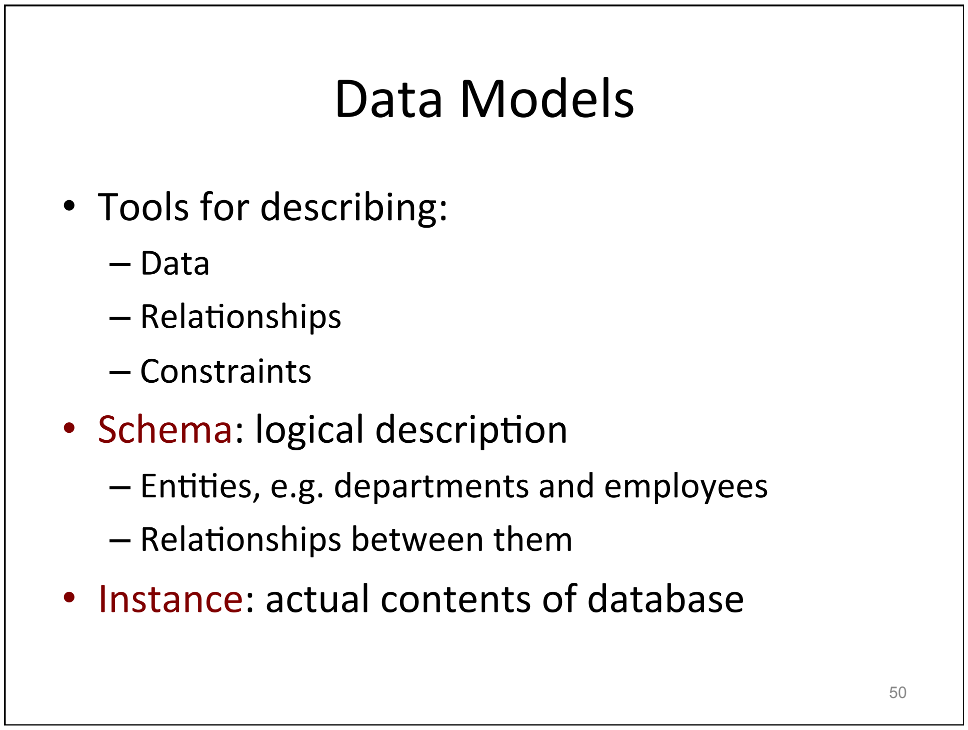


Motivation for database systems

- 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).
```

49



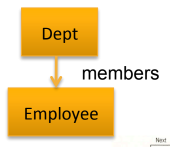
Data Models

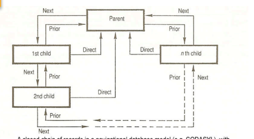
- Tools for describing:
  - Data
  - Relationships
  - Constraints
- Schema: logical description
  - Entities, e.g. departments and employees
  - Relationships between them
- Instance: actual contents of database

50

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”



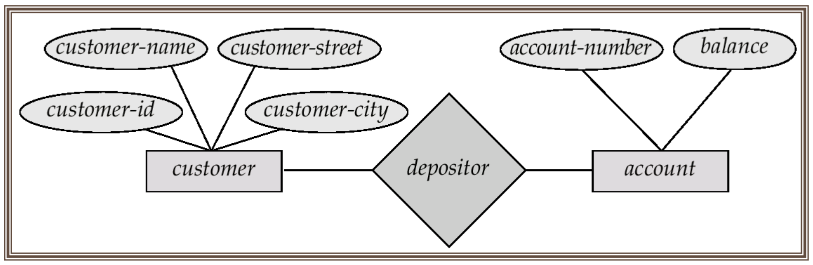


A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

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

51

Entity-Relationship Data Model



52

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

53

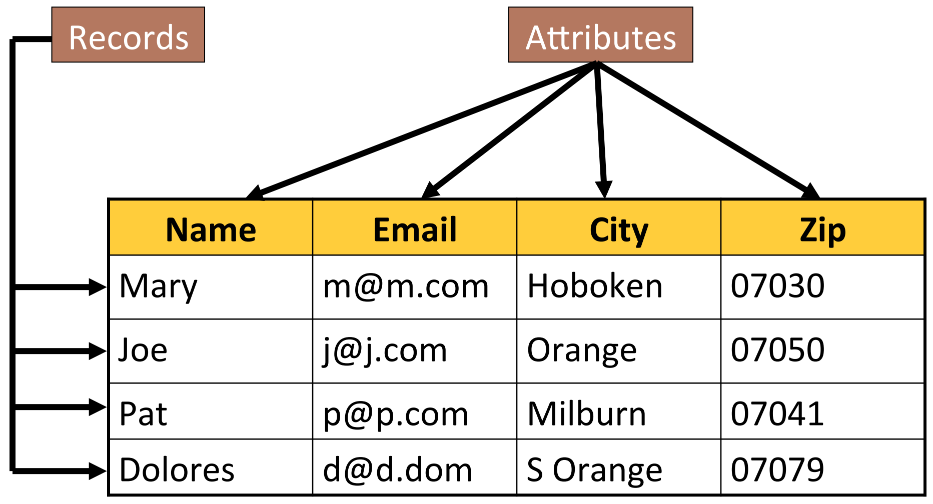
RELATIONAL DATA MODEL

54

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

57

Identity Columns

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

58

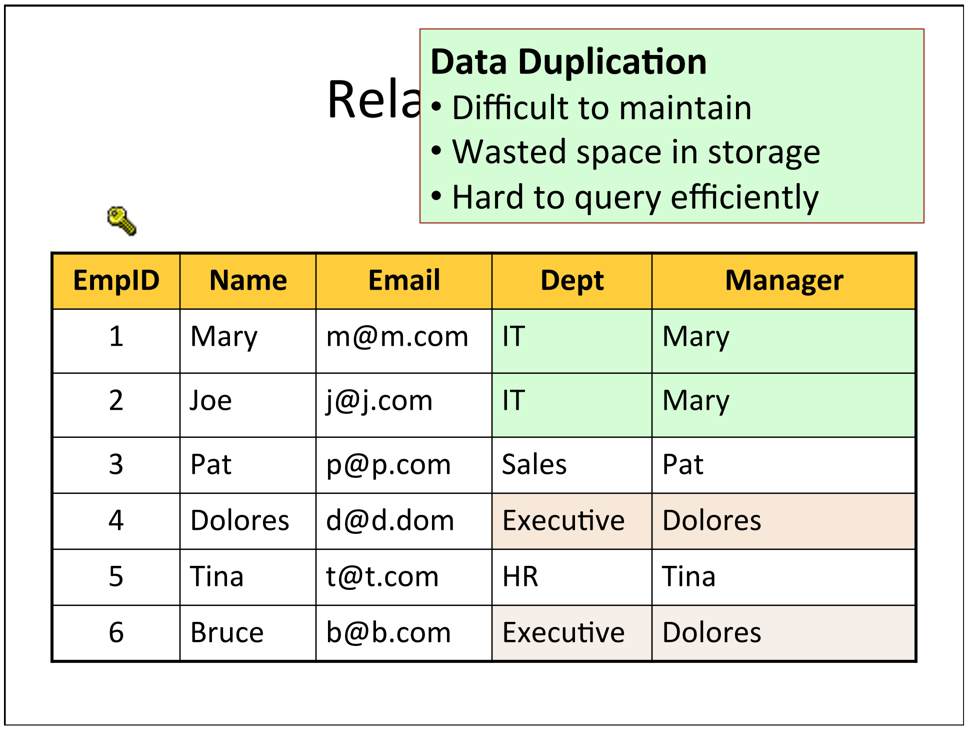
Identity Columns

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

59

RELATIONSHIPS IN THE RELATIONAL MODEL

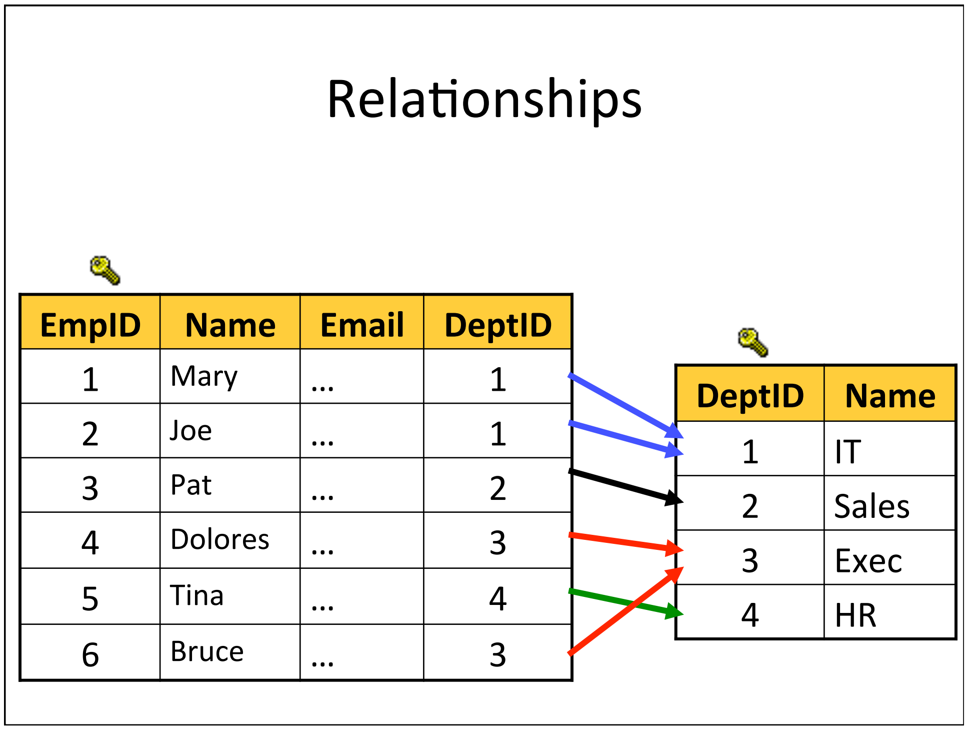
60



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

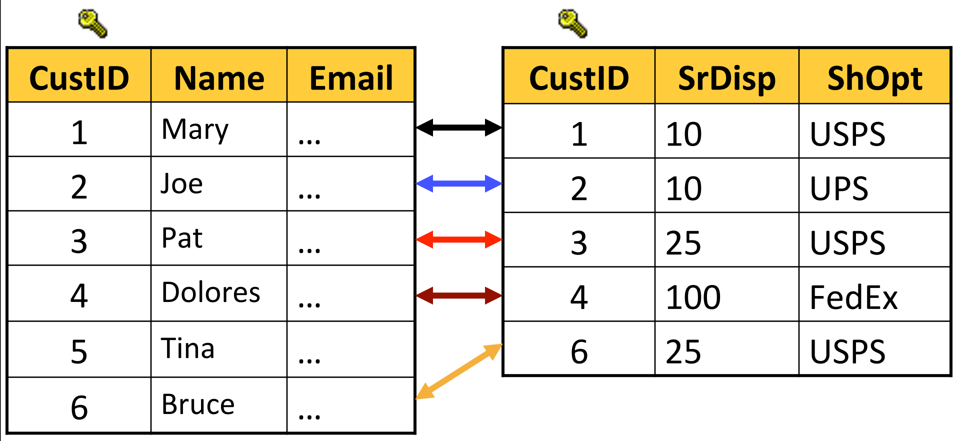
DeptID	Name
1	IT
2	Sales
3	Exec
4	HR

Relationships

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

63

One-to-one



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

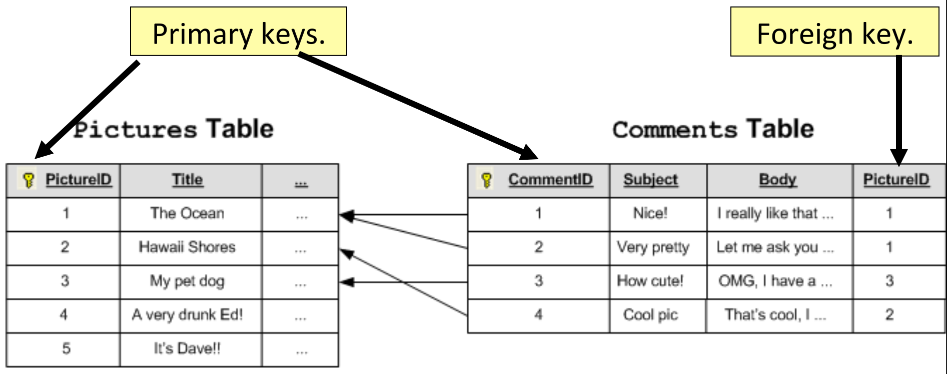
CustID	SrDisp	ShOpt
1	10	USPS
2	10	UPS
3	25	USPS
4	100	FedEx
6	25	USPS

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

65

One-to-many



Pictures Table

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

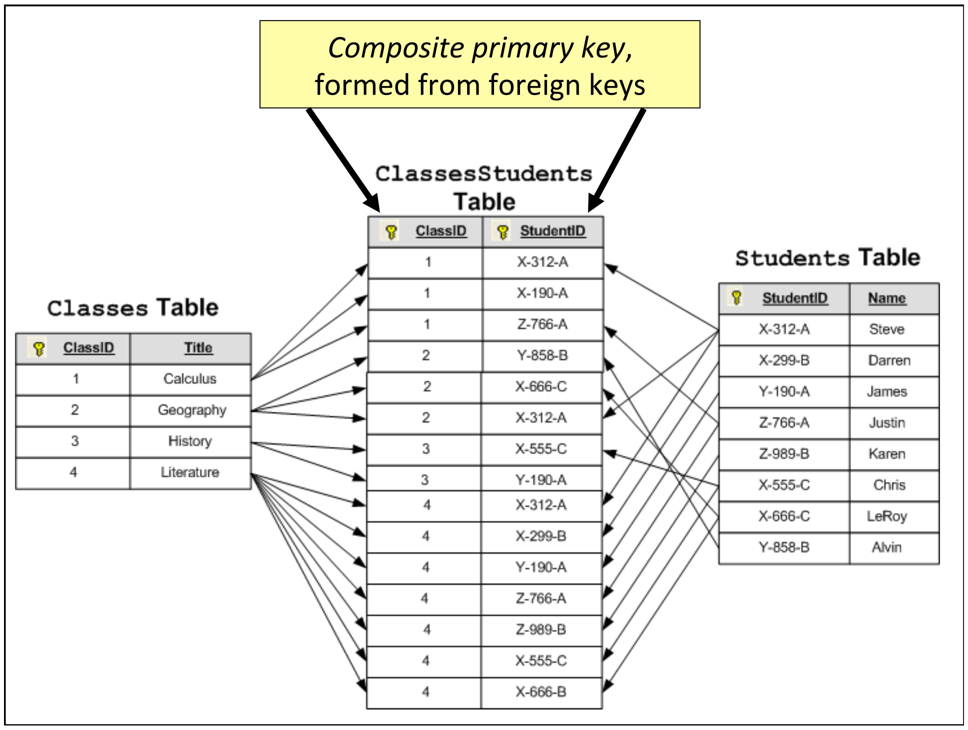
Comments Table

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

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

67



Classes Table

ClassID	Title
1	Calculus
2	Geography
3	History
4	Literature

ClassesStudents Table

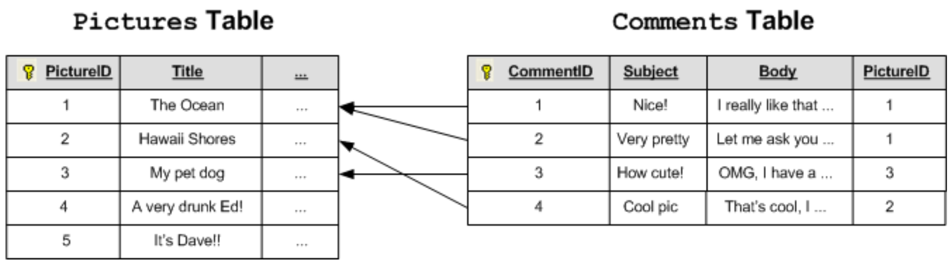
ClassID	StudentID
1	X-312-A
1	X-190-A
1	Z-766-A
2	Y-858-B
2	X-666-C
2	X-312-A
3	X-555-C
3	Y-190-A
4	X-312-A
4	X-299-B
4	Y-190-A
4	Z-766-A
4	Z-989-B
4	X-555-C
4	X-666-B

Students Table

StudentID	Name
X-312-A	Steve
X-299-B	Darren
Y-190-A	James
Z-766-A	Justin
Z-989-B	Karen
X-555-C	Chris
X-666-C	LeRoy
Y-858-B	Alvin

Referential Integrity

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



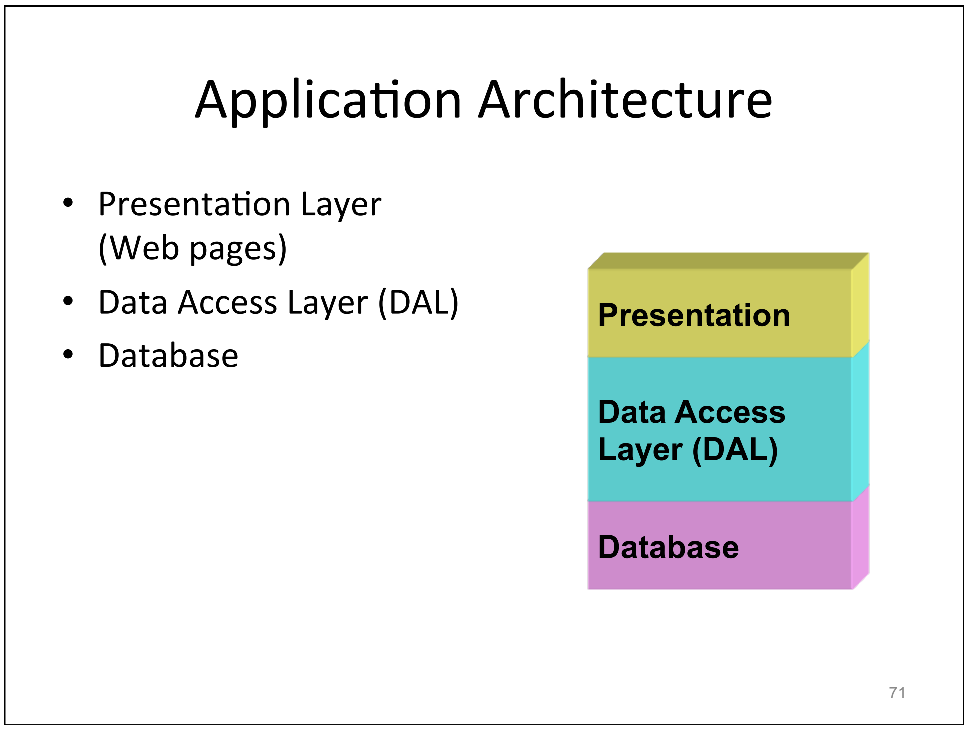
Pictures Table

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

Comments Table

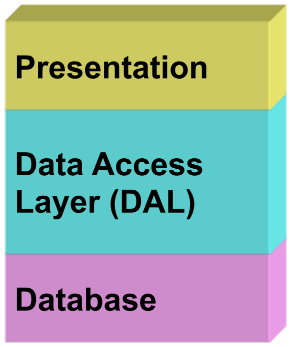
CommentID	Subject	Body	PictureID
1	Nice!	I really like that ...	1
2	Very pretty	Let me ask you ...	1
3	How cute!	OMG, I have a ...	3
4	Cool pic	That's cool, I ...	2

APPLICATION ARCHITECTURE

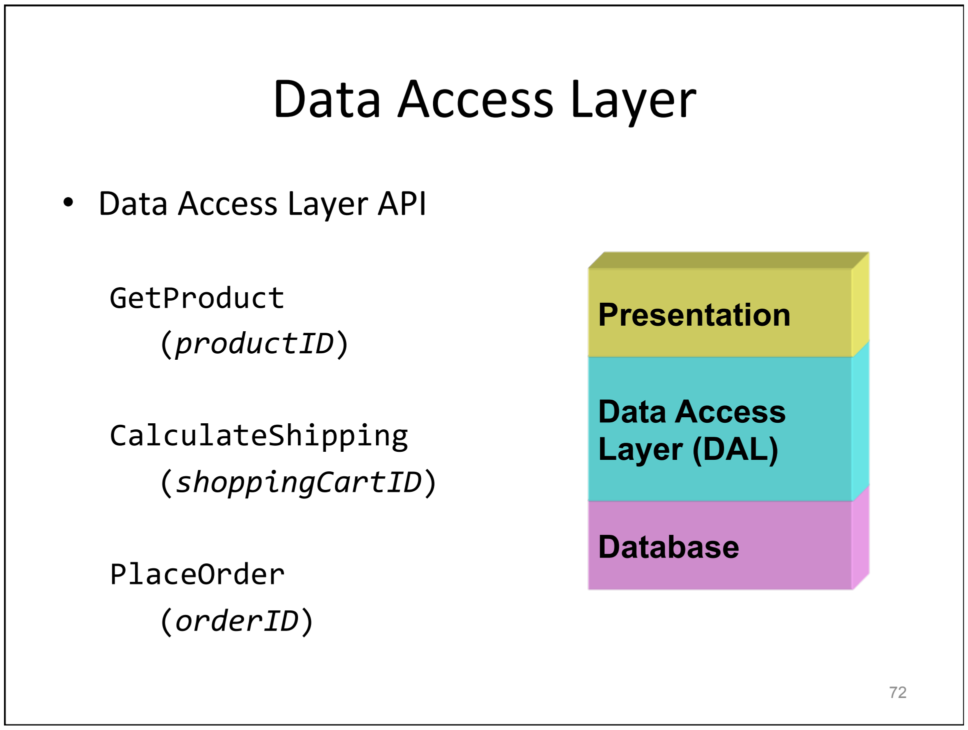


Application Architecture

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



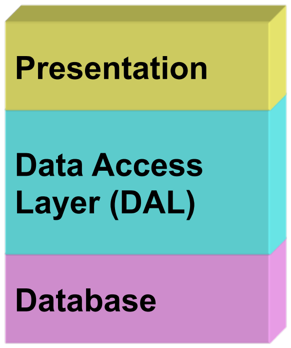
71



Data Access Layer

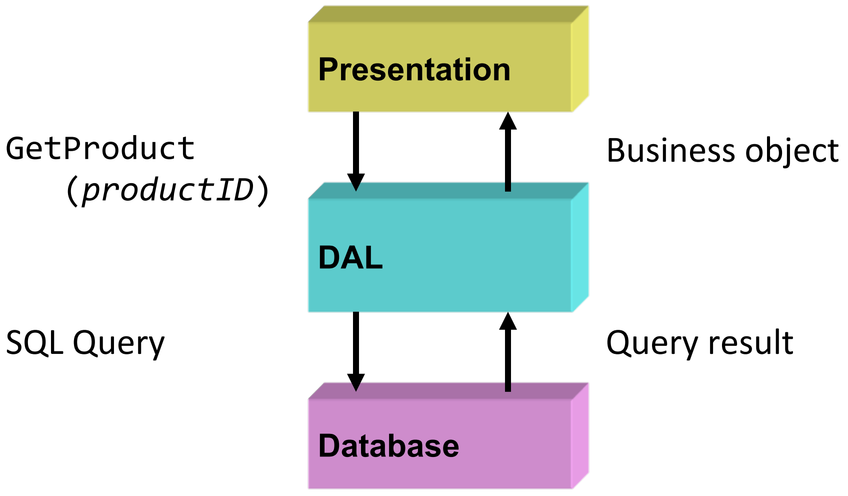
- Data Access Layer API

```
GetProduct
    (productID)

CalculateShipping
    (shoppingCartID)

PlaceOrder
    (orderID)
```



72

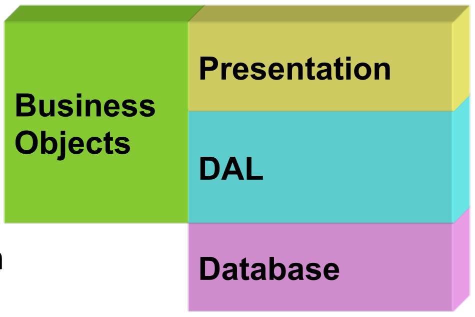
DAL in Architecture



73

Business Objects

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

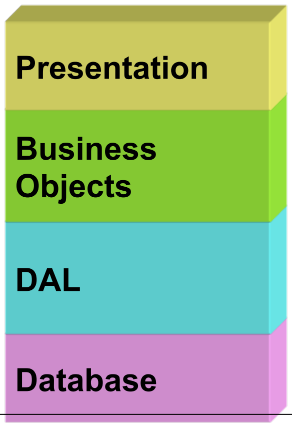


Example code:

```
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

76

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

77

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

78

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

79

ENTITY FRAMEWORK CODE FIRST

80

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

81

EF Code First

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

82

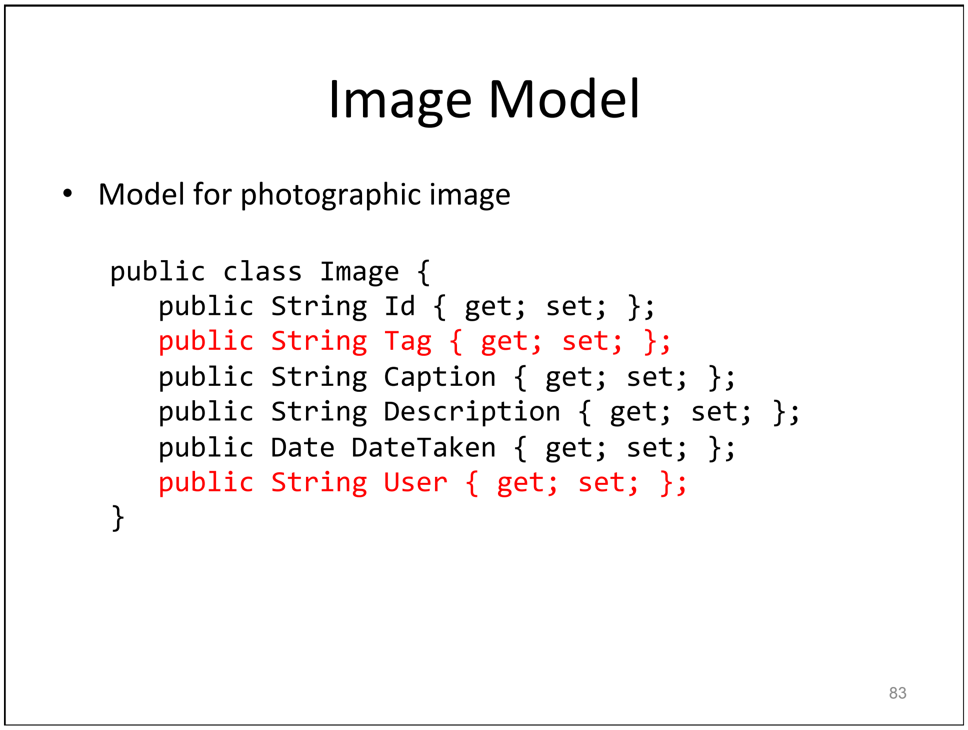


Image Model

- 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; };
}
```

83

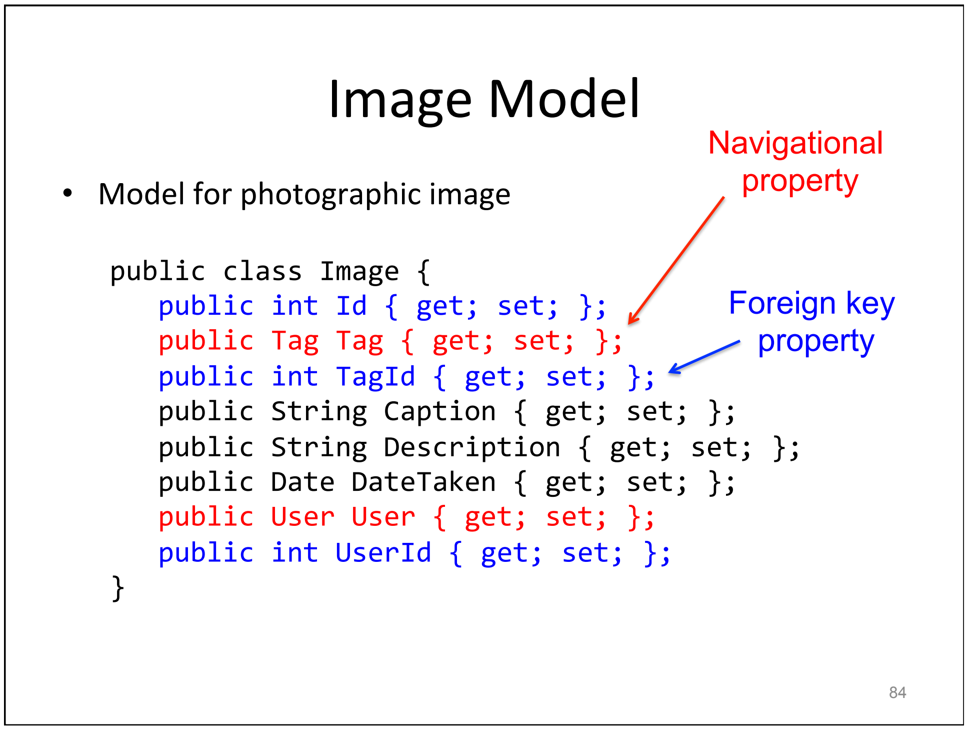


Image Model

- Model for photographic image

```
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; };
}
```

Navigational property

Foreign key property

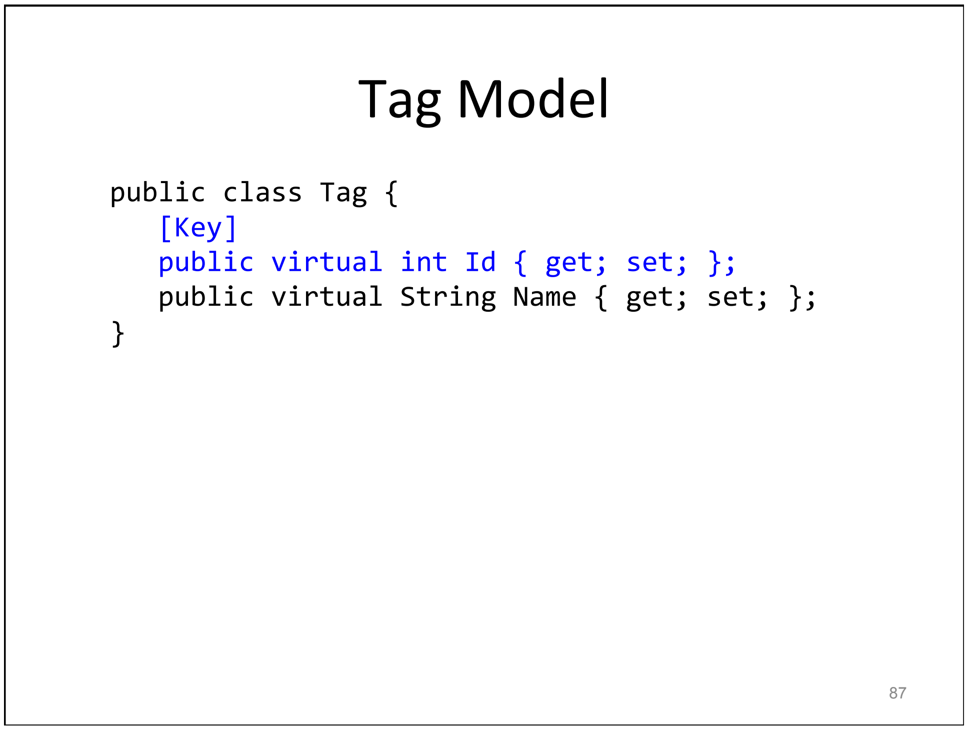
84

Image Model

```
public class Image {
   [Key]
   public int Id { get; set; };
   public Tag Tag { get; set; };
   [ForeignKey]
   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]
   public int UserId { get; set; };
}
```

Image Model

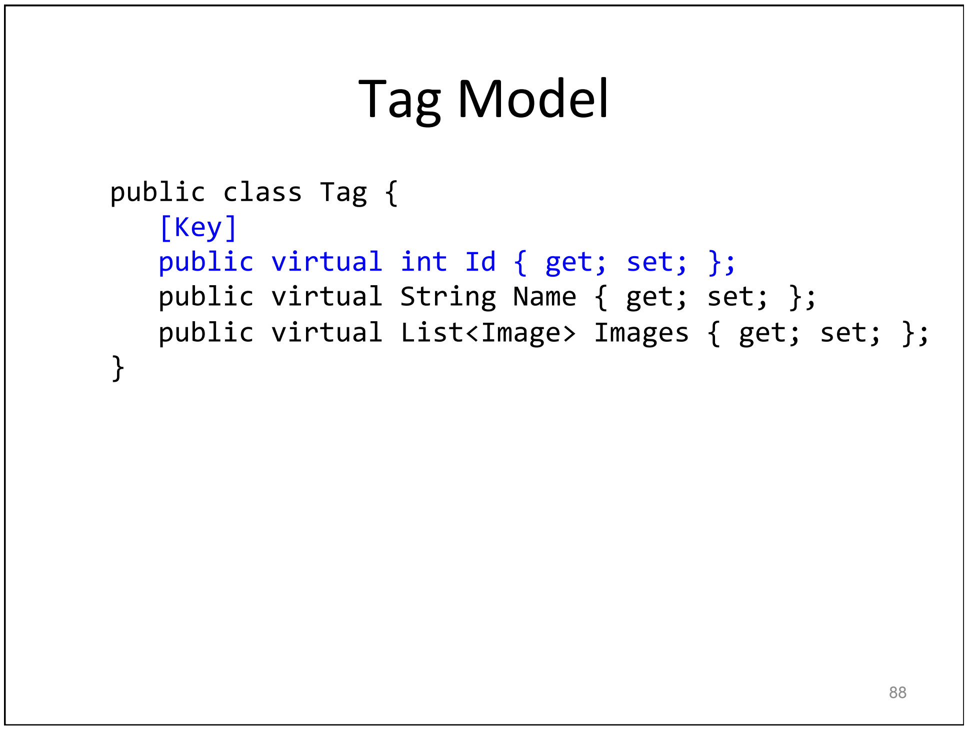
```
public class Image {
   [Key]
   public virtual int Id { get; set; };
   public virtual Tag Tag { get; set; };
   [ForeignKey]
   public virtual int TagId { get; set; };
   public virtual String Caption { get; set; };
   public virtual String Description
                          { get; set; };
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



Tag Model

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

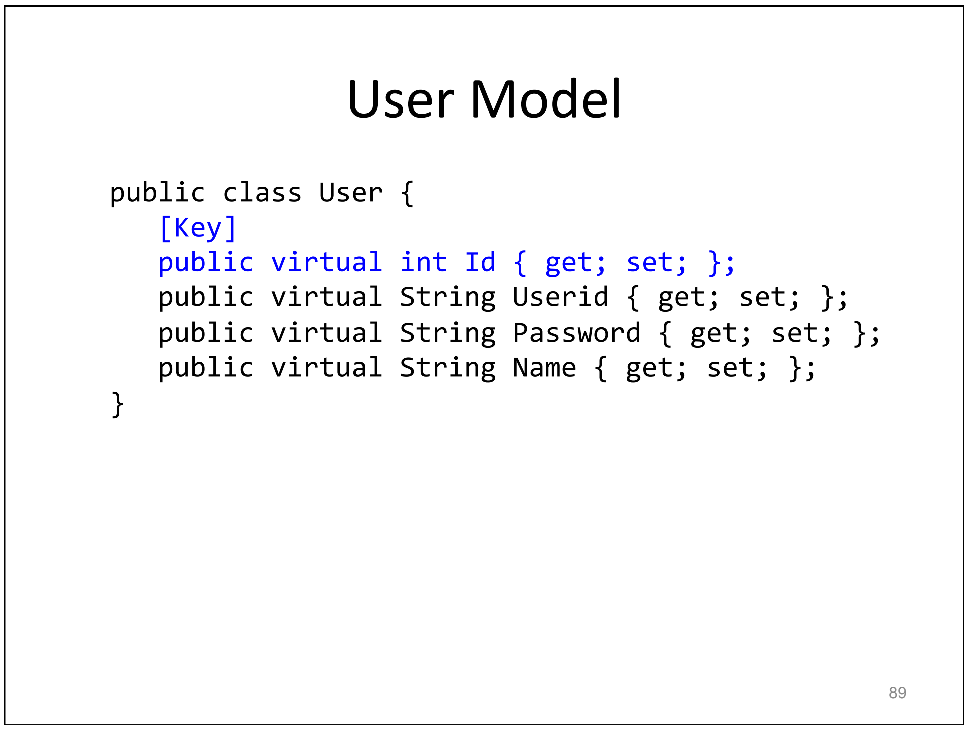
87



Tag Model

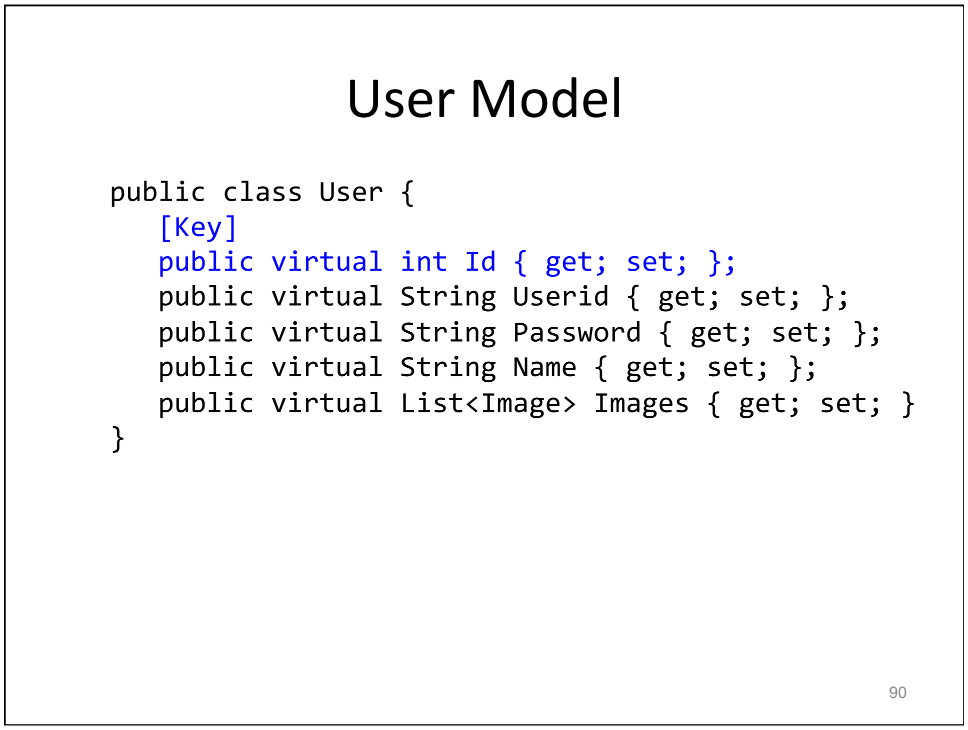
```
public class Tag {
   [Key]
   public virtual int Id { get; set; };
   public virtual String Name { get; set; };
   public virtual List<Image> Images { get; set; };
}
```

88



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; };
}
```



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; }
}
```

ACCESSING THE DATABASE

Database Context

- Database connection session:

```
using System.Data.Entity;

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

Lazy vs Eager Loading

- Lazy loading (N+1 problem):

```
var db = new ImageSharingDB();
var images = db.Images;
```

- Eager loading:

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

93

Querying the Database

- LINQ query

```
var db = new ImageSharingDB();
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> @Html.ActionLink("Create New", "Create") </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>@Html.DisplayFor(m ⇒ item.Caption)</td>
  <td>@Html.DisplayFor(m ⇒ item.Tag)</td>
  <td>@Html.DisplayFor(m ⇒ item.User)</td>
  <td>
    @Html.ActionLink("Edit", "Edit",
                     new {id=Image.Id})
    @Html.ActionLink("Details", "Details",
                     new {id=Image.Id})
    @Html.ActionLink("Delete", "Delete",
                     new {id=Image.Id})
  </td>
</tr> }
```

96

Database Connection

- Connection String (`web.config`)

```
<connectionStrings>
   <add name="ImageSharingDB"
      connectionString=
          "data source =(localdb)\v11.0;
           Integrated Security=SSPI;
           initial catalog=ImageSharing"
      providerName="System.Data.SqlClient" />
</connectionStrings>
```

- Default Connection

97

Database Initializers

- Default: init on start or on model change
- Specify in global.asax.cs:

```
protected void Application_Start() {
  Database.SetInitializer(
    new DropCreateDatabaseAlways
      <ImageSharingDB>());

  AreaRegistration.RegisterAllAreas();
  RegisterGlobalFilters (GlobalFilters.Filters);
  RegisterRoutes( RouteTable.Routes);
}
```

98

Seeding the Database (1/2)

```
Public class ImageSharingDBInitializer
  : DropCreateDatabaseAlways<ImageSharingDB> {

  protected override void Seed(ImageSharingDB db) {
    db.Users.Add(new User {Userid="johndoe",...});

    db.Tags.Add(new Tag {Name="architecture"});

    db.Images.Add(new Image {
          User=new User{Userid="joeblow",...},
          Tag=new Tag{Name="landscape",...},
          Caption=..., Description=...});
    base.Seed(db);
}
```

99

Seeding the Database (2/2)

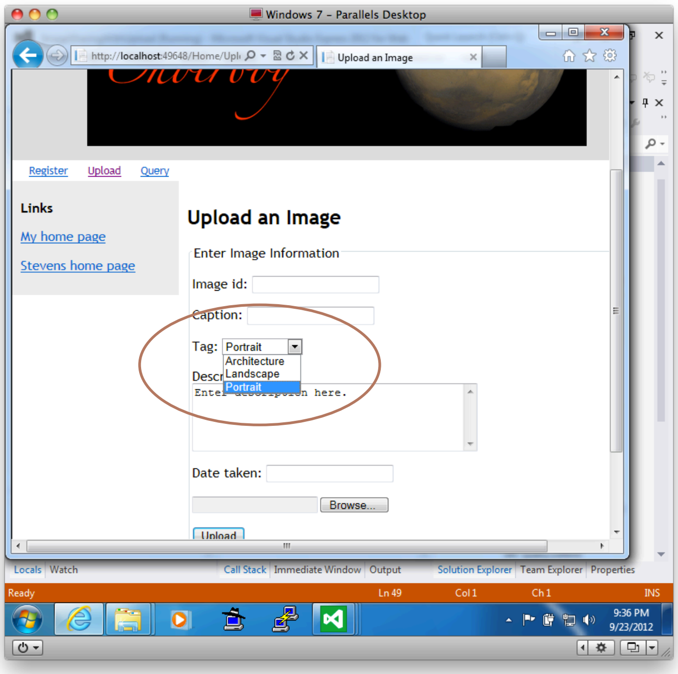
- Specify in global.asax.cs:

```
protected void Application_Start() {
  Database.SetInitializer(
    new ImageSharingDbInitializer());

  AreaRegistration.RegisterAllAreas();
  RegisterGlobalFilters (GlobalFilters.Filters);
  RegisterRoutes( RouteTable.Routes);
}
```

100

EDIT ACTION

101



Upload an Image

Enter Image Information

Image id:

Caption:

Tag: Portrait

- Architecture
- Landscape
- Portrait

Descr

Enter description here.

Date taken:

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; };
}
```

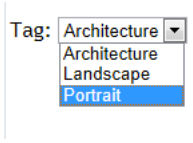
- Database of tags

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

ListItem

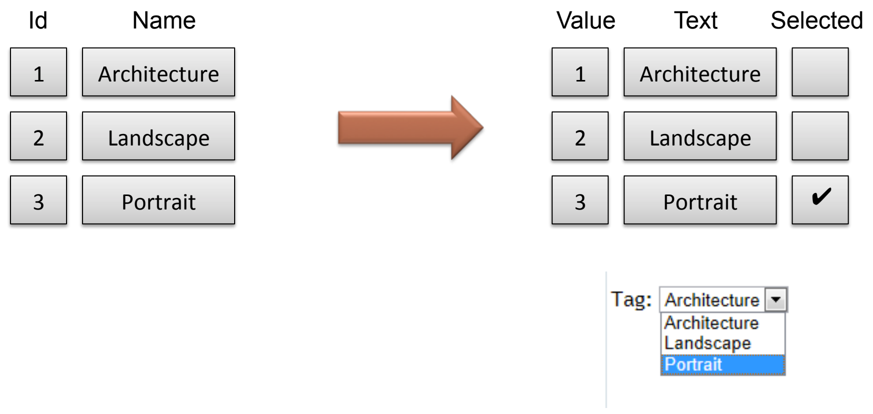
Value	Text	Selected
1	Architecture	
2	Landscape	
3	Portrait	✔

- 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; }
}
```

SelectList Helper



Id	Name
1	Architecture
2	Landscape
3	Portrait

Value	Text	Selected
1	Architecture	
2	Landscape	
3	Portrait	✔

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

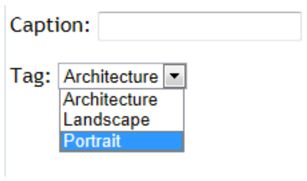
Id	Name	Selected
1	Architecture	
2	Landscape	✔
3	Portrait	

- 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

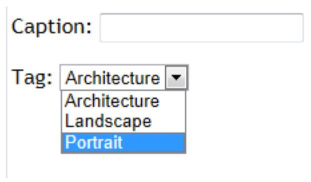

Caption:

Tag: Architecture

```
@{ Html.BeginForm(...); }
  <fieldset>
    <legend> Edit Image</legend>
    <p> @Html.LabelFor(m ⇒ m.TagId)
        @Html.DropDownListFor(m ⇒ m.TagId,
           ViewBag.Tags as SelectList) </p>

    <p> @Html.LabelFor(m ⇒ m.Caption)
        @Html.EditorFor(m ⇒ m.Caption)
        @Html.ValidationMessageFor
           (m ⇒ m.Caption) </p>

    <input type=“submit" value=“Save" />
  </fieldset>
@{ Html.EndForm(); }
```

108

Edit View


Caption:

Tag: Architecture

Architecture
Landscape
Portrait

```
@{ Html.BeginForm(...); }
  <fieldset>
    <legend> Edit Image</legend>
    <p> @Html.LabelFor(m ⇒ m.TagId)
        @Html.DropDownListFor(m ⇒ m.TagId,
           ViewBag.Tags as SelectList) </p>

    <p> @Html.LabelFor(m ⇒ m.Caption)
        @Html.EditorFor(m ⇒ m.Caption)
        @Html.ValidationMessageFor
           (m ⇒ m.Caption) </p>

    <input type="submit" value="Save" />
  </fieldset>
@{ Html.EndForm(); }
```

109

Edit View

Value	Text	Selected
1	Architecture	
2	Landscape	✔
3	Portrait	

```
@{ Html.BeginForm(...); }
  <fieldset>
    <legend> Edit Image</legend>
    <p> @Html.LabelFor(m ⇒ m.TagId)
        @Html.DropDownListFor(m ⇒ m.TagId,
           ViewBag.Tags as SelectList) </p>

    <p> @Html.LabelFor(m ⇒ m.Caption)
        @Html.EditorFor(m ⇒ m.Caption)
        @Html.ValidationMessageFor
           (m ⇒ m.Caption) </p>

    <input type="submit" value="Save" />
  </fieldset>
@{ Html.EndForm(); }
```

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);
}
```

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);
}
```

Edit View

```
@{ Html.BeginForm(...); }
  <fieldset>
    <legend> Edit Image</legend>

    <p> @Html.LabelFor
            (m ⇒ m.ImageToEdit.TagId)
        @Html.DropDownListFor
            (m ⇒ m.ImageToEdit.TagId,
             m ⇒ m.Tags) </p>
    ...
    <input type="submit" value="Save" />
  </fieldset>
@{ Html.EndForm(); }
```

113

Edit Form

```
<form action="/Image/Edit/17" method="post">
  <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);
}
```

Edit Processor

```
[HttpPost]
Public ActionResult Edit(int id,
                         FormCollection collection) {
   Image image = ImageSharingDb.Images.Find(id);
   TryUpdateModel(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);
}
```