**ER DIAGRAMS**

**Entity:** is an abstract object of some sort. Entity sets are a collection of similar entities.

**Relationship:** an association among two or more entities. A collection of similar relationships forms a relationship set. Draw **relationship sets** as **diamonds**

**Attributes**: Both entity sets and relationship sets can have properties or attributes. Drawn as ovals.

**Keys:** The key of an entity set is an attribute, or a minimal collection of attributes, that uniquely identify entities in t hat set. Keys are underlined.

**Key Constraints:** each member of a entity set participates in at most one relationship in a relationship set.

**Many-to-one:** E.g. In the relationship set “Makes” joining “Directors” and “Movies”, each Movie has at most one director. Draw an **arrow** from “Makes” to “Directors”

**Many-to-many:** StarsIn is a many-to-many relationship set. Each star can participate in many movies, and each movie can have many stars.

**Referential Integrity Constraints:** If in a many-to-one relationship set from entity set E to entity set F, there must be exactly one entity in F for each one in F (rather than at most one), then this is a referential integrity constraint. Indicate this by drawing a rounded arrowhead. **E.g. instead of an arrow** draw a **rounded arrow** for the “Movies”/”Director” many-to-one relation.

**Class Hierarchies:** subcategories of entity sets. E.g. Movies has subcategories of “Animated” and “Live Action”. Use an “ISA” triangle to represent the subcategories.

**Weak Entity Set:** no attribute(s) of an entity set can uniquely identify an entity – need a key of another entity to uniquely identify it. Say that the other entity is the identifying owner. E.g. Film crews identified by number, but may have film crews of same number at different studios. There must be a referential integrity constraint from the weak entity set to its owner (each weak entity has exactly a single owner). Identify these by drawing a double box around the weak entity set and a Double diamond for the relationship connecting it to its owner.

**FROM ER DIAGRAMS TO SCHEMA**

**Key Constraints to Schema:** E.g. Movies(Title, Year, Length, Director.Name, YearMade) – take all the attributes and put them in movies

**Class Hierarchies to Schema:** E.g. Movie(Title, Year, Length) Type Animated(Title, Year, Computer Animated) Live Action(Title, Year)

**Weak Entity Sets to Schema:** E.g. FilmCrew(Number, Name) Studio(Name) – Don’t include the foreign key

**Properties of ER Diagrams:** **Should be faithful** (i.e. represent the data needed), **Should be simple** (no extra unneeded entities or relationships), **Should avoid redundancy** (i.e. don’t keep the same data in multiple places)

Each relationship uniquely identified by the keys of the entities that it joins. So can’t have two or more relationships with the same keys (e.g. employee ssn and dept.) Make another entity Duration with attributes “from” and “to”

**DB NORMALIZATION**

**Kinds of Normal Form:** 1NF, 2NF, 3NF, Boyce-Codd Normal Form, 4NF, 5NF

**First Normal Form:** if each tuple in a relation has at most one value per attribute (i.e. can’t have two rows with the same name).

**Functional Dependency:** We say that attributes B1, …, Bk are functionally dependent on A1, …, An if given the values of A1,…, An, each Bi can have only one value.

**Full Functional Dependency:** We say that B1, …, Bk are fully functionally dependent on A1, …, An if A1, …, An is a minimal set of attributes such that A1, …, An ! B1, …, Bk.

**2NF:** it is in 1NF and every non-primary-key attribute in a relation is fully functionally dependent on that relation’s primary key.

**Transitive Dependence:** The attributes C1, …, Cm are transitively dependent on A1, …, An via B1, …, Bk if A1, …, An ! B1, …, Bk and B1, …, Bk ! C1, …, Cm .E.g. In Orders(OrderID, OrderDate, CustomerID, CustomerName,

CustomerAddress) we have OrderID ! CustomerID and CustomerID ! CustomerName so that CustomerName is transitively dependent on OrderID via CustomerID.

**3NF:** it is in 2NF and has no non-primary key attribute that is transitively dependent on the primary key.

**BCNF:** whenever A1,…, An !B1,…,Bk in a relation, then A1,…, An is a superkey for the relation.

**TRANSACTIONS**

**ACID Transactions:** **Atomic:** Either all the steps of a transaction happen or none of them do. **Consistent:** Transact ions leave the DB in a consistent state. **Isolated:** Transactions execute independently of one another. **Durable:** A successfully completed transaction is permanently recorded in the DB.

**TRANSACTIONS IN SQL**

By default, mysqli queries commit every query. To **start a transaction** we need to turn this off. $mysqli->autocommit(FALSE); Then carry out the steps of the transaction, and either

**commit** or **rollback**. $mysqli->commit(); or $mysqli->rollback(); **then**

$mysqli->autocommit(TRUE);

**PREPARED STATEMENTS**

One way to get around SQL injections

$query = “SELECT \* FROM CalendarEntries WHERE username = ? AND date = ?” $stmt = $mysqli->stmt\_init(); if ($stmt->prepare($query)) { $stmt->bind\_param(‘ss’, $\_POST[‘username’], $\_POST[‘date’]); $stmt->execute(); } while ($row = $stmt->fetch\_row()) {

… }

**Parameters for bind\_param:**

s – string, i – integer, d – double, b – binary

**Prepared statements ensure**: inputs are properly quoted and only the intended SQL is executed

**PDO (PHP DATA OBJECTS)**

Built in class to handle calls to DBs.

**PDO Methods:**

$mypdo = new PDO(“mysql: host=localhost; dbname=Movies db”, “root”, “\*\*\*\*\*\*”);

This is a **constructor method** that returns a PDO object connecting to a MySQL DB.

$pdoobj->exec(string)

Executes SQL query string and returns

number of affected rows

$pdoobj->query(string)

Executes SQL query string and returns a

“PDO Statement”

E.g. $result = $mypdo->query(“SELECT \* FROM Movies”);

$PDOstatement->fetch()

Returns next row of the table as an array

E.g. while ($row = $result->fetch()) {

print(“<td>”.$row[‘ Tit le’].“</td>”);

… }

**Shortcut for iterating through the results:** $query = “SELECT \* FROM Movies”; foreach ($mypdo->query($query) as $row) { print(“<td>”.$row[‘ Title’].“</td>”); … }

**Prepared Statements in PDO:** $query = “SELECT \* FROM CalendarEntries WHERE username = ? AND date = ?” $stmt = $mypdo->prepare($query); $stmt->bindParam(1,$\_POST[‘username’]); $stmt->bindParam(2,$\_POST[‘date’]); $stmt->execute(); while ($row = $stmt->fetch()) { … }

**Transactions in PDO:**

$mypdo->beginTransaction();

**to start the transaction**

$mypdo->commit();

**to commit the current transaction**

$mypdo->rollback();

**to rollback the current transaction**

**Reason for Using PDOs:** we can easily change the database we’re working with.

**SQLite:** another DBMS (like MySQL), except that the DB gets stored in a file (which is then easy to copy from one machine to another).

**The only change** to the code you would need to make to use SQLite instead is the **initial connection**: $mypdo = new PDO(“sqlite:filename”); **instead o**f $mypdo = new PDO(“mysql:host=hostname;dbname=dbname”, username, password)

**Other DBMS:** PostgreSQL and MS SQL Server

**XML (eXtensible Markup Language)**

XHTML a special case of XML Data elements are marked by tags; enclosed by starting tag <tagname> and ending tag </tagname>. Can also have tags that don’t enclose data, in which case we add a slash to close them: <tagname /> Tags are properly nested. Tags can have attributes. E.g. <tagname attr1=“value1” attr2=“value2”> Can define our own tags: hence extensible. Also, there is a root element: a tag that starts and ends the entire document.

**DTD (Document Type Declaration):** specifies information about what tags and attributes are allowable. If we specify the **DTD at the top of the document**, it can be used to **validate** that the following XML conforms to that DTD.

E.G. <!DOCTYPE booklist SYSTEM “booklist.dtd”> - “booklist.dtd is the file containing the validating DTD

**RSS (REALLY SIMPLE SYNDICATION):**

RSS is a means of publishing blog entries, news items, etc. so that they can be read in an aggregator. (I.e. items come to you, you don’t go looking for them).

**Basic Format:** The root tag is <rss>. Within this, the basic tag is <channel>. The most common children tags of <channel> are:

• <title>: The name of the feed or site

• <link>: The URL of the site associated with the feed

• <description>: A brief description of the site

**Example:** **<item> <title>**The Life of the Party**</title><link>**http://weblog.fortnow.com/2006/04/life-of-party.html**</link> <description>** From Jay Leno's monologue on Monday's Tonight Show **<blockquote>** Scientists have been working on a device that will tell when you are boring or irritating in social situations. Who really needs this device?&hellip;Scientists. **</blockquote>** Normally I'd complain about such stereotypes, but social grace is just not one of our strengths**.</description></item>**

**Write Your Own RSS Feed:**

$fp = fopen(“./info230. xml”, "w"); fputs($fp,"<?xml version=\"1.0\"?>\n"); fputs($fp,"<rss version=\"2.0\">\n"); fputs($fp,"<channel>\n"); fputs($fp,"<title>INFO/CS 2300: Intermediate Web Programming and Design</title>\n"); fputs($fp,"<link>http://info2300.cs.cornell.edu</link>\n");

fputs($fp,"<description>Announcements about INFO/CS

2300 Spring 2009</description>\n");

$query = "SELECT content, DATE\_FORMAT(created on, \"%a, %d %b %Y %T EST\") as time, createdby FROM announcements ORDER BY createdon

DESC";

$result = $mysqli->query($query); while ($row = $result->fetch\_assoc()) { fputs($fp,"<item>\n"); fputs($fp,"<title>INFO/CS 2300 Announcement</title>\n"); fputs($fp,"<link>http://info2300.cs.cornell.edu</link>\n"); fputs($fp,"<description>"); fputs($fp,$row['content']); fputs($fp,"</description>\n"); fputs($fp,"<author> (");

fputs($fp, $row[‘createdby’]); fputs($fp,") </author>\n"); fputs($fp,"<pubDate>"); fputs($fp,$row['time']); fputs($fp,"</pubDate>\n"); fputs($fp,"</item>\n"); } fputs($fp,"</channel>\n"); fputs($fp,"</rss>\n"); fclose($fp);

**Reading RSS Feeds:** Use Simple XML

$xml = simplexml\_load\_file("http://info2300.cs.cornell.edu/info230.xml");print("<h2>".$xml->channel->title."</h2>\n"); foreach ($xml->channel->item as $item) { print("<h3><a href=\"".$item->link."\">".$item->title."</a></h3>\n"); print("<p>".$item->description."<br/>\n"); print("Posted by ".$item->author."<br/>\n"); print("Posted on ".$item->pubDate."</p>\n"); print("<hr/>\n"); }

**XML and Ajax:** in the JavaScript call back function we can access the result in

request.responseXML instead of

request.responseText

**JSON (JavaScript Object Notation)**

JSON is a useful way of transmitting data for web applications. Is just like JavaScript

{“booklist”: [ { “genre”: “fantasy”, “format”: “hardcover”, “author”: { “firstname”: “Neal”,

“lastname”: “Stephenson” },

“title”: “Anathem”, “published”: 2008 },

{ “genre”: “science fiction”, “author”: { “firstname”: “Orson”, “middlename”: “Scott”, “lastname”: “Card” }, “title”: “Ender’s Game”, “published”: 1985 } ] }

**Accessing Info in JSON:** Using JavaScript, we can easily navigate

information in JSON.

var json = {“booklist”: … } document.writeln(json.booklist[1].author.firstname); document.writeln(json.booklist[0].title);

**JSON and Ajax:** The benefit: we can take the text returned from the Ajax call, evaluate it, and start navigating the arrays as described earlier. j son = eval ( ‘(‘ + request.responseText + ‘)’ );

document.writeln(json.booklist[1].author.firstname); document.writeln(json.booklist[0].title);

**JSON Parsing:** Safer to have the data parsed first to make sure

it is just a JSON array. E.g. json = JSON.parse(request.responseText);

**JQUERY**

**Common Javascript Libraries:** prototype, scriptaculous, dojo, mochikit, yahoo ui, google web toolkit, moo tools, pyjamas, jquery.

**How to Begin Using jquery:** download jquery.js from jquery.com, move it inside your project (say, under /js) link to it in the <head> of your HTML: <script type=”text/javascript” src=”js/jquery.js”></script>

**Why jquey:** concise and very readable, CSS selectors (and XPath operators), chaining, automatic looping, plugins, very small file size, excellent community and collection of tutorials books!

**Example:** $(document).ready(function() {

$("a").click(function() { alert("Hello world! "); }); });

**jquery selectors: $(), example:** d $(‘a’)

$(‘a,p,span’), **$(‘#my\_div’) vs document.getElementById(‘ my\_div’),** $(‘input:not(:checked) + span’), $(“input[name=’newsletter’]”), $(“div:contains(‘john’)”), $(‘div:has(p)’)

**jquery Chaining:** Can string together a whole bunch of function calls E.g. $('p.surprise').addClass('ohmy').show('slow') or $(this).parent().find(‘#person\_email’).val()

**Automatic looping:** E.g. $(‘p.inbox’).each(function() {

$(this).fadeIn(); });

**jquery Plugin Example:** table-sorting plugin

Jquery shares: you can use other libraries too

**SUBVERSION AND DEPLOYMENT**

Subversion allows you to achieve the Copy-Modify-merge model. Allows you to keep track of every change made to the files. If you want to go back to a previous version, you can.

**Using version control software: svn status**

Get a list of all the changes you’ve made t o your local version

**svn revert filename**

Undo the changes to a file you made

**svn commit**

Commit all your local changes to the repository

**svn update**

Get current version of the code from the repository

**TortoiseSVN**  is an example of an SVN client

**Version Control Etiquette:** Don’t commit code that is not working

**Deployment:** How to create your own website: 1. Find a hosting ser vice. 2. Register a domain name. 3. Upload your code and DB.

**FRAMEWORK**

**Pros: Once** you know a framework well , you can quickly put sites together

**Cons: Large** learning curve and **Most** frameworks want you to do things ‘their way’. May inhibit development in

some cases.

**Available in different languages:** Java, Perl, Python (Dj ango, TurboGears), PHP (CakePHP, symfony, Zend, …), Ruby (Ruby on Rails)

**MODEL-VIEW-CONTROLLER**

**Model:** DB/object definitions, **View:** HTML + PHP to show data from model, **Controller:** Event handling to deal with changes to model , direct new displays of views.

**ORM (OBJECT RELATIONAL MAPPING)**

DB => Code

Table => Object

Row => Instance

Attribute => Field

Then objects have methods like ‘save’ and ‘find’ to store/retrieve rows from the DB.

**SAMPLE BLOG**

After downloading CakePHP, we have a file layout that looks like this: **config:** database.php, routes.php **models:** post.php\* **controllers:** posts\_controller.php\* **views** posts\*: index.ctp\*, view.ctp\*, add.ctp\*, edit.ctp\* layouts: default.ctp\* Where “\*” means we create these files

**Database Configuration:** class DATABASE\_CONFIG {

var $default = array( 'driver' => 'mysql', 'persistent' => false, 'host' => 'localhost', 'login' => 'cakeblog', 'password' =>'letseatcake', 'database' => 'cakeblog',

'prefix' => '', ); }

**The Model:** <?php class Post extends AppModel { var $name = 'Post'; } ?>

**Routing:** The routes/dispatching part of the framework automatically takes a URL

and figures out which action to call in

which controller, and which view to

show (render). **E.g**. cakeblog/posts/index

calls the ‘index’ action in the posts controller and renders the ‘ index’ view **E.g.** cakeblog/posts/edit/3 calls the ‘edit’ action in the posts controller and renders the ‘edit’ view.

**Controller: index:** Example of an act ion in

app/controllers/posts\_controller.php:

function index() { $this->set('posts', $this->Post->find('all')); } sets a variable called ‘posts’ for use in the view to be all posts in the DB.

**Controller: action:** function view($id = null) { $this->Post->id = $id; $this->set('post', $this->Post->read()); } takes the id from the URL, pulls t he appropriate

post from the DB, and sends it to the view in the variable ‘post’.

**Controller: add:** function add() {

if (!empty($this->data)) { if ($this->Post->save($this->data)) { $this->Session->setFlash("Your post has been saved."); $this->redirect(array(' action' => 'index'));} } }

**Data Validation:** We can add to our model file information about what constitutes a valid instance of a post.

<?php class Post extends AppModel { var $name = 'Post'; var $validate = array( 'title' => array( 'rule' => 'notEmpty' ), 'body' => array( 'rule' => 'notEmpty' ) ); } ?>

No SQL. All the DB access has been

hidden away.

• Substantially shorter code: forms in a

few short lines!

• Separation between coding logic and

display logic.