Advanced Web Development

Siddharth Kaza
Towson University, Computer and Information Sciences

Outline

- Problems in Lab 1? Other issues?
- Models and persistent storage in web applications
- ORM Demo
- Presentation
- Project ideas?
- Project proposal
  - Due next week
- ORM Lab

Persistent storage in web applications

- Most interesting applications store data
  - Content Provision
    - News, Weather, Magazines
    - Other "content-driven" sites
  - Commerce
    - Catalog, Orders
    - Customers
  - Community participation
    - Tagging, photos, social networks, blogs+responses...

Persistent storage = RDBMS?

- Other models exist
  - Flat files
  - XML databases – eXist-db
  - RDF
  - Amazon SimpleDB
  - Microsoft Azure

All valid possibilities, but relational databases are still the "bread-and-butter" of many domains and applications.

Why RDBMS?

- Minimize redundancy
  - SQL: powerful, standard query language
- Mature technology with strong implementations
  - Lots of legacy data
- Flexible
  - Hierarchical and other models can be simulated in relational framework.

What other issues with RDBMS?

- Speed?
- Understandability?
- Maintainability?
- Mapping to OO?

RDBMS with Ruby on Rails

- Default: SQLLite http://www.sqlite.org/
  - Embedded DB in the form of a C library – used in firefox, android, various others..
- PostgreSQL: http://postgresql.org
- Both open-source, freely downloadable, run on Unix/Linux, OS X, Windows, etc.
- Others: DB2, Frontbase, SQL Server, MySQL, Sybase, Oracle

DB support for applications (mysql)

- mysqladmin –-user=... --password=.. create dbname
- HeidiSQL - www.heidisql.com - gui front end
- Add tables
- Define relations, etc
- Write SQL to get data out.
- This is how things traditionally have been done.

PERL example

source: http://inconnu.isu.edu/~ink/perl_cgi/lesson3/database.html

```
#!/usr/bin/perl
use strict;
use DBI;
use CGI;
# setup database connection variables
my $user = "some username goes here";
my $password = "some password goes here";
my $host = "the host you want to connect to";
my $driver = "mysql";
# connect to database
my $dsn = "DBI:$driver:database=cis430;host=$host";
my $dbh = DBI->connect($dsn, $user, $password);
# setup CGI handle
my $cgi = new CGI;
# start HTML
.....
```

PERL example (cont.)

```
print $cgi->header . $cgi->start_html('Some Tunes');
# handle any queries that have been sent our way
my $artist = validate($cgi->param('Artist'));
my $song = validate($cgi->param('Song'));
my $album = validate($cgi->param('Album'));
if ($cgi->param('Query')) {
    my $sql = "select filename, artist, album, track, song from mp3s "
    $sql .= "where artist like '$artist' "
    if (length $artist > 1);
    $sql .= "where song like '$song' "
    if (length $song > 1);
    $sql .= "where album like '$album' "
    if (length $album > 1);
    $sql .= "order by artist, album, track";
    my $rows = $dbh->selectall_arrayref($sql) || die $dbh->errstr;
```

PERL example (cont.)

```
if (@$rows) {
  print "<table border=1 cellspacing=0 cellpadding=3><tr>" .
  "<th>Filename</th><th>Artist</th><th>Album</th><th>Track</th><th>Song</th><
  /tr>";
  foreach my $row (@$rows) {
  print "<tr><td>" . join ("</td><td>", @$row) . "</td></tr>\n";
}
print "</table>\n";
}
else {
  print "<p><i>No matches found</i></p>\n";
}

}
........
print $cgi->end_html();
# disconnect from database
$dbh->disconnect();
exit(0);
```

What's wrong with these examples (flashback to lecture 1)

- General concerns with these approaches
  - Specifics of database tied directly in
  - What happens if we change databases, or even database servers?
- SQL embedded in the code
  - Code depends on data model
- Design lesson: add indirection to reduce coupling to specifics of current configuration
  - Data model defined "outside" of the application – in DDL language for database
    - Would like to tie things together more cleanly.

Data model problems

- Look at the data model

```
if ($cgi->param('Query')) {
my $sql = "select filename, artist, album,
track, song from mp3s ";
```

- One table, three columns: artist, song, album, track
- Artist name,album repeated for each track on an album.
  - Not very normal-form-ish
- Should probably be 3 tables (at least)
  - Lots of joins, - or multiple queries

2 strategies for relations

1. Joins between tables
   - Select from artists, albums, songs where album.artist_id=artists.artist_id and song.album_id = album.album_id and artist.artist_id =..

2. Sequence of queries: use first to lead you to second
   - Find the Id of an artist,
   - Find the albums for that artist
   - Find songs

- Both make sense at different time points, but...
  - Neither fits with how we'd like to program in today's OO languages

Objects vs. Relations

- Relational databases:
  - Relations and joins are key
  - Get a set of results and iterate over it.
  - Join tables or run new queries to "walk" data relations

- Object Oriented Development
  - Each concept is an object
  - Relationships indicated by containment, is-a, has-a, etc.
  - Use methods to find related data
  - Inheritance of attributes and methods

Find albums and songs for each artist

- Relational: SQL

```
select from artists, albums, songs where
album.artist_id=artists.artist_id and
song.album_id =
album.album_id and artist.artist_id =..
```

- Object-Oriented

```
Artist a = artistList.findArtist(name);
Albums = a.getAlbums();
Collection songs;
for each album in Albums {
    songs.add(album.songs);
}
```

Object-Relational Translation

- If we are to use relational data storage together with object-oriented programming languages, we need a way to translate between the two.

- Can be done manually
  - Each class (with helpers) has attributes and methods that can be used to populate instances based on SQL Queries.
  - Use these classes to read things in as necessary.

- Or, we don't translate at all and use OO Databases
  - That option is not very practical, RDBMS most prevalent

Shortcomings of Object-Relational Translation

- Potentially RDBMS dependent.
- Ties code to details of data model
- Pushes need for management of database towards the application
  - As opposed to being in the middle.
- Does not evolve with the data model.
  - Need new getters/setters

Shortcomings cont. – configuration issues

- Hard-wiring database name and location into code is problematic
- Need to rebuild code with every change to the DB
- One solution - configuration files specify dbname and location
  - Helpful, but only to a point.
  - What if I want to change not only DB machine, but RDBMS also?

So what do we need?

1. Easy translation between relational and data models
2. Data model described in one place (not several)
3. Ability to evolve data model without large amounts of new code associated with each change
4. Optimized data access without large amounts of hand tweaking
5. Ease of configuration

First the configuration issues

- Use abstract data access model to hide differences between RDBMS systems
  - Common API reduces (but may not eliminate) difficulties of switching
    - JDBC
    - ODBC
- Use APIs and configuration tools to standardize location and types of data sources.

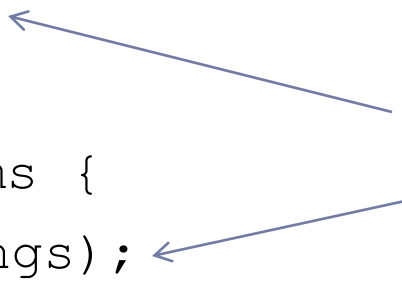
Object-Relational Mapping

- Automated, configurable translation between data models and object-oriented languages
- Create, retrieve, update, and delete (CRUD) data records without using SQL
- Objects are instantiated as needed without programmer intervention.
- Implicit navigation of relational links.

OO code with ORM

- Accessors present implicit queries

```
Artist a = artistList.findArtist(name);
Albums = a.getAlbums();
Collection songs;
for each album in Albums {
    songs.add(album.songs);
}
```



ORM in Web frameworks

- Hibernate- http://www.hibernate.org
  - J2EE
- MyBatis: http://blog.mybatis.org/
  - Java, .net, Ruby/Rails
- Entity Framework
  - .Net
- Ruby on Rails – built-in
  - But maybe not as powerful/flexible as others.

ORM in Rails

- ActiveRecord – main class
- Migrations - define database structures
- Models – use OO code to validate and do other processing.
- Accessors/Mutators inferred by reflection:
  - System observes its own behavior to get things done
  - Models provide methods that ask about contents of tables

ORM in Rails example:

- Lets create the application first
- Create the database (if using mysql or other):
  - `Mysqladmin –u root create orm_development`
  - Or use a GUI: www.heidisql.com / PHPAdmin
- See and understand the config file:
  - config/database.yml
  - Don't embed connection information in code
- Check the config: `rake db:migrate`
  - Rake is the rails version of make

Creating tables

- Either we can do it manually using `create table` or `GUI`. There are drawbacks :
  - History of all the changes are lost
  - The changes have to be made both at development and production databases
  - Saving SQL scripts and versioning is a problem

- Rails
  - Migrations
    - Allows you to create tables, alter table, add data without writing SQL
    - Manages versions
    - Allows easy replication and synchronization across database servers

Migrations

- `rails generate model product`

- See the db/migrate folder for the scripts
  - The 'up' method
    - Making the changes
  - The 'down' method
    - Undoing the changes
  - The up and down are combined as the 'change'
  - A nice cheat sheet of what can be done inside migration
    - http://dizzy.co.uk/ruby_on_rails/cheatsheets/rails-migrations

Rails ORM demo/lab

- First Project – Lec3ORMDemo
  - This adds a course table to mini application we did on listing courses for instructors
  - We learn about
    - Models, views, controllers, instance variables, rake, migrations, working in terminal and IDE

- Lab (on blackboard)

Presentation and discussion

Project teams – TO DO

To Do –

1. Use blackboard groups for communication. It can be setup to email you.

1. Version control repositories
    1. Either use google groups or,
    2. bitbucket

How many of you have used SVN before?

How many of you have used Git before?

Project proposal content

- The overall aim of the project
- Major classes of objects that will be needed
- Use cases (does not have to diagrams)
  - Who will be the users?
  - How will they interact with the system?
- Other additional features authentication, security, deployment, mashups, mobile interfaces, etc.
- Platform
- Group members
- Not more than 2 pages

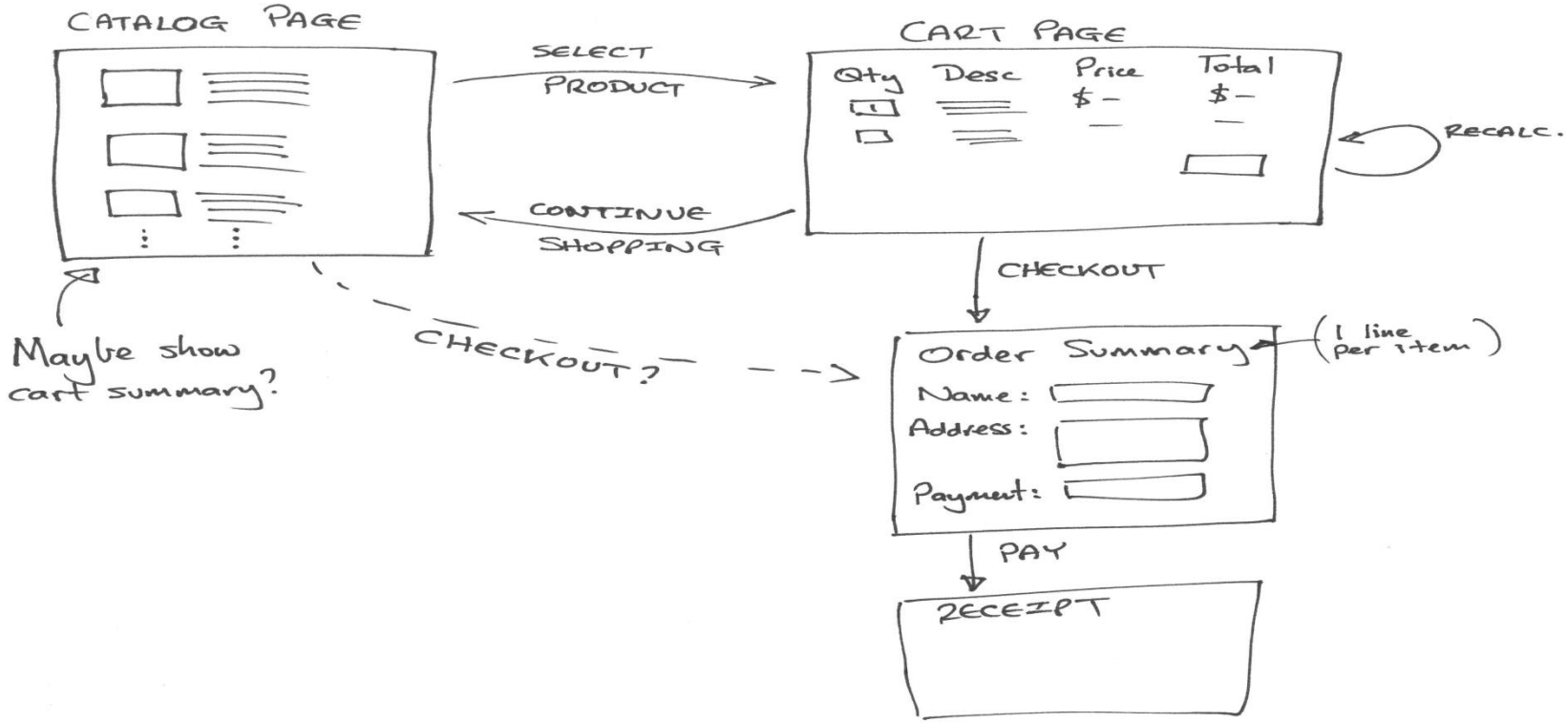
Let's do an example (scaled – down) project proposal

- Aim: Design an online book store
- Major classes
- Uses cases:
  - Scenarios, stories, page flows

Major classes

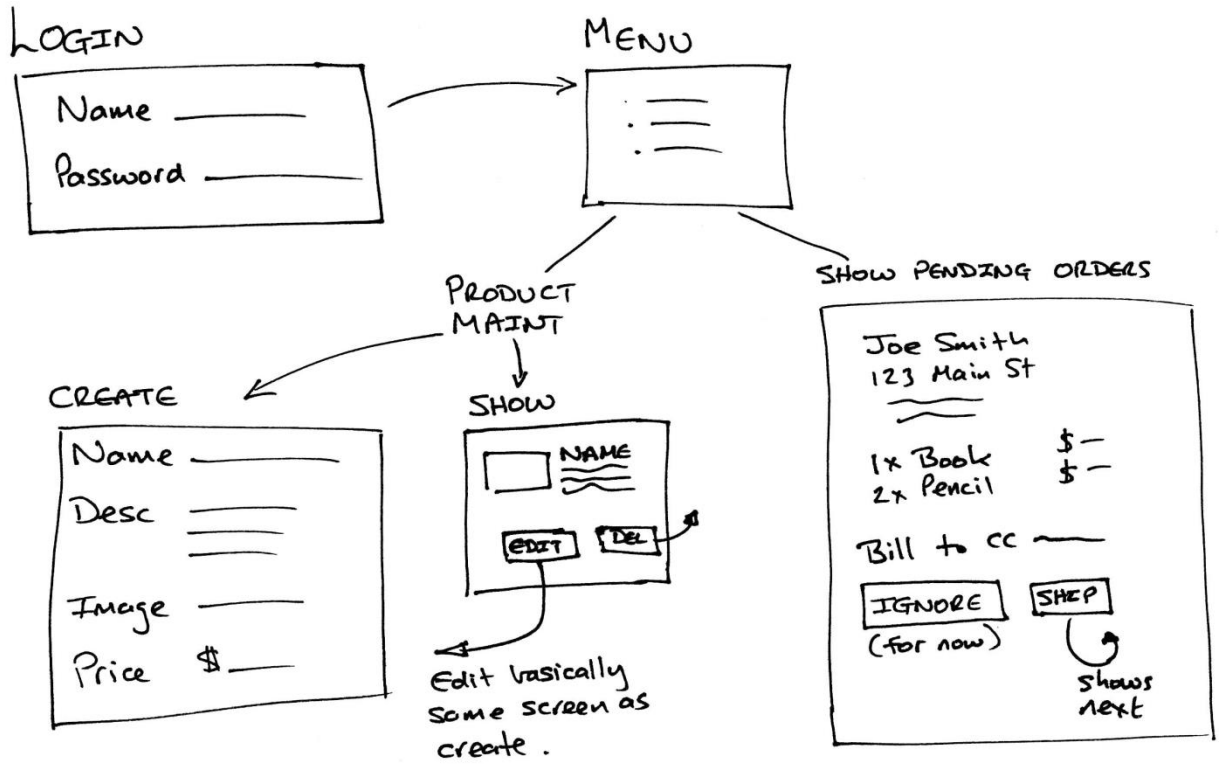
- Books
- Buyer
- Seller
- Cart
- Invoice
- Etc..

The Buyer Pages



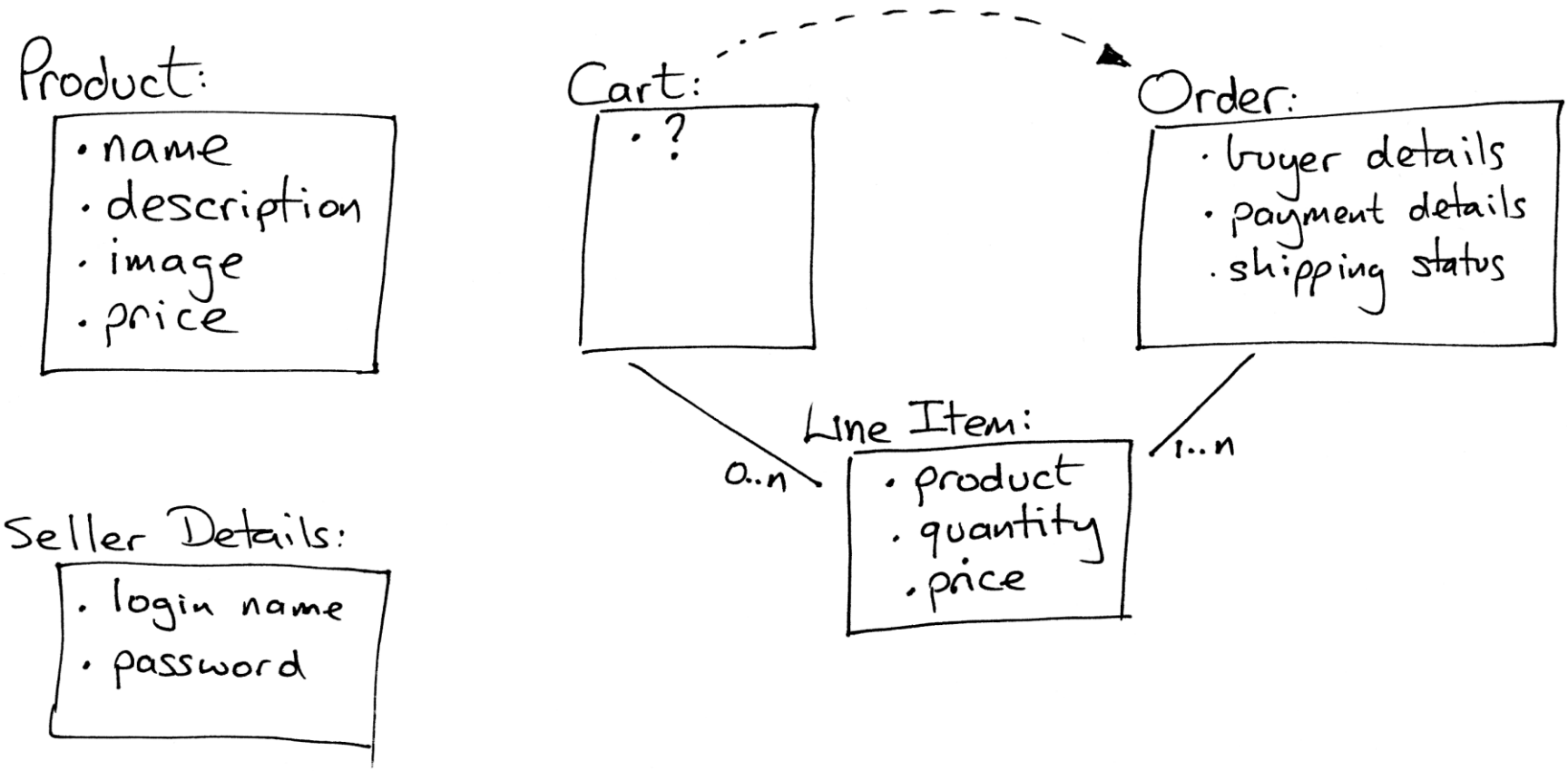
Flow of buyer pages

The seller pages



Flow of seller pages

Data


Product:
• name
• description
• image
• price
Cart:
• ?
Order:
• buyer details
• payment details
• shipping status
Line Item:
• product
• quantity
• price
0..n
1..n
Seller Details:
• login name
• password