

PHP and MySQL Best Practices

Luke Welling

luke@omniti.com

<http://lukewelling.com>

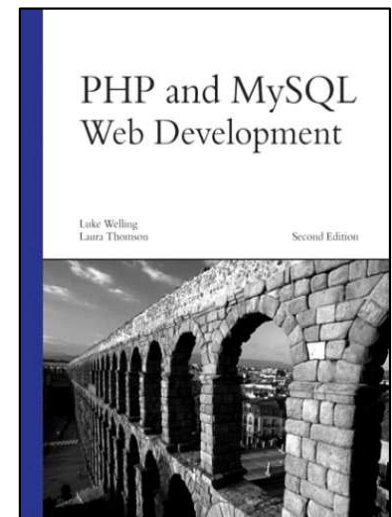
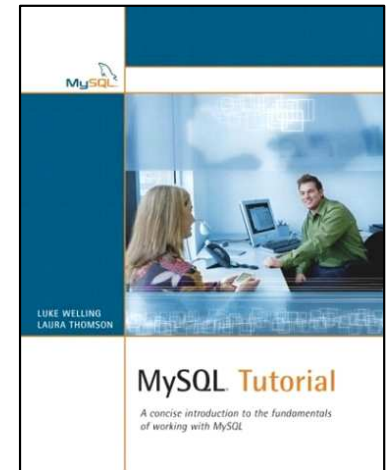
Laura Thomson

laura@laurathomson.com

<http://www.laurathomson.com>

About the Speakers

- Used MySQL and PHP since last century
- More than a decade of web development experience in a range of languages, using a range of databases



- Many web developers use PHP and MySQL
- PHP and MySQL are easy to use and you can teach yourself the basics very quickly
- This always seems to happen in project crunch mode
- Learning about the more interesting (and useful) parts of PHP and MySQL happens on a demand basis
- Learning best practices often happens as a result of making mistakes
- This tutorial is really a series of small tutorials on how to use PHP and MySQL well and about interesting parts of PHP and MySQL

- Tell me about you
- This material was written for people who use PHP but do not consider themselves experts
- Participation is encouraged
- We can select subtopics according to what you are interested in

- **General:**
 - Good practices for PHP and MySQL projects
 - A small rant about project management
- **PHP topics:**
 - Frameworks
 - Database connectivity
 - PHP OO traps
 - Security
 - Performance tuning
- **MySQL topics**
 - Storage engines
 - Query caching
 - Full text searching
 - Security
 - Performance tuning
 - Replication
- **On demand topics:**
 - More on OO, transactions, foreign keys, subqueries, stored procedures...

Best, or at Least Good, Practices

- By best practices, I mean:
 - Knowing what's in the toolbox
 - Knowing the strengths and weaknesses of your tools
 - Using the tools appropriately, and in the most efficient fashion
 - Having a working, consistent, repeatable process.

- The process needs managing
 - Use a version control system
 - Have a testing and release process
 - Have an easy way for developers to set up a staging server
 - Plan for scale

- Subversion, CVS, Bitkeeper, Perforce, Visual Source Safe ...
- Store everything possible
 - Code
 - Text
 - Images (including raw and base or stock library)
 - Configuration files
 - Documentation
- As well as all the usual attributes of version control that apply to other software projects, there could come a day when you need to be able to say with confidence exactly what your website was promising on a day in the past

- Because errors in web text are easy to fix when noticed, many people get lazy
- At the very least you need a staging server where content can be reviewed and approved, and a live server. (And for anything other than the most microsite you should have at least dev/stage/prod)
- For simple sites they could be on the same install, but that makes it impossible to test new software configurations, versions and installs. (Virtualization can help you here.)
- Somebody other than the original author should review all text and test all code

Write good code

- Above all, code needs to be readable and maintainable
- Using this week's cool design pattern, obscure performance tweaks, or niche language features are probably not positive attributes
- Clever code is not always good code
- If you look back at your own code from 6 months ago and think "Wow, I must be really smart to have understood that then," it is a bad sign

- `E_STRICT` and `error_reporting`
- Exception handling, the good and the bad
- Unit testing
- Debugging: `apd`, `xdebug`, `kcachegrind` (to be covered later, in the section on Performance Tuning)

Error reporting

- Turn `error_reporting` up on your dev server – `E_STRICT` is something to aim for at the moment. (PHP 6 will be different)
- Fix any reported issues even if they are not problems.
- Turn `display_errors` off on your production server – gives attackers too much information

Exception handling

- PHP 5's exception handling is irritating, because not all extensions use it
- Use `set_error_handler()` and `set_exception_handler()` for top level errors
- Whacking all your code in a `try...catch` block is not a panacea.

- Good tools available:
 - PHPUnit: <http://phpunit.sourceforge.net/>
 - test_more: bundled with Apache-Test
<http://search.cpan.org/dist/Apache-Test/>
- Difficulty is not the tools but the discipline (and the time)

Web Development Project Management (I promise this won't take long.)

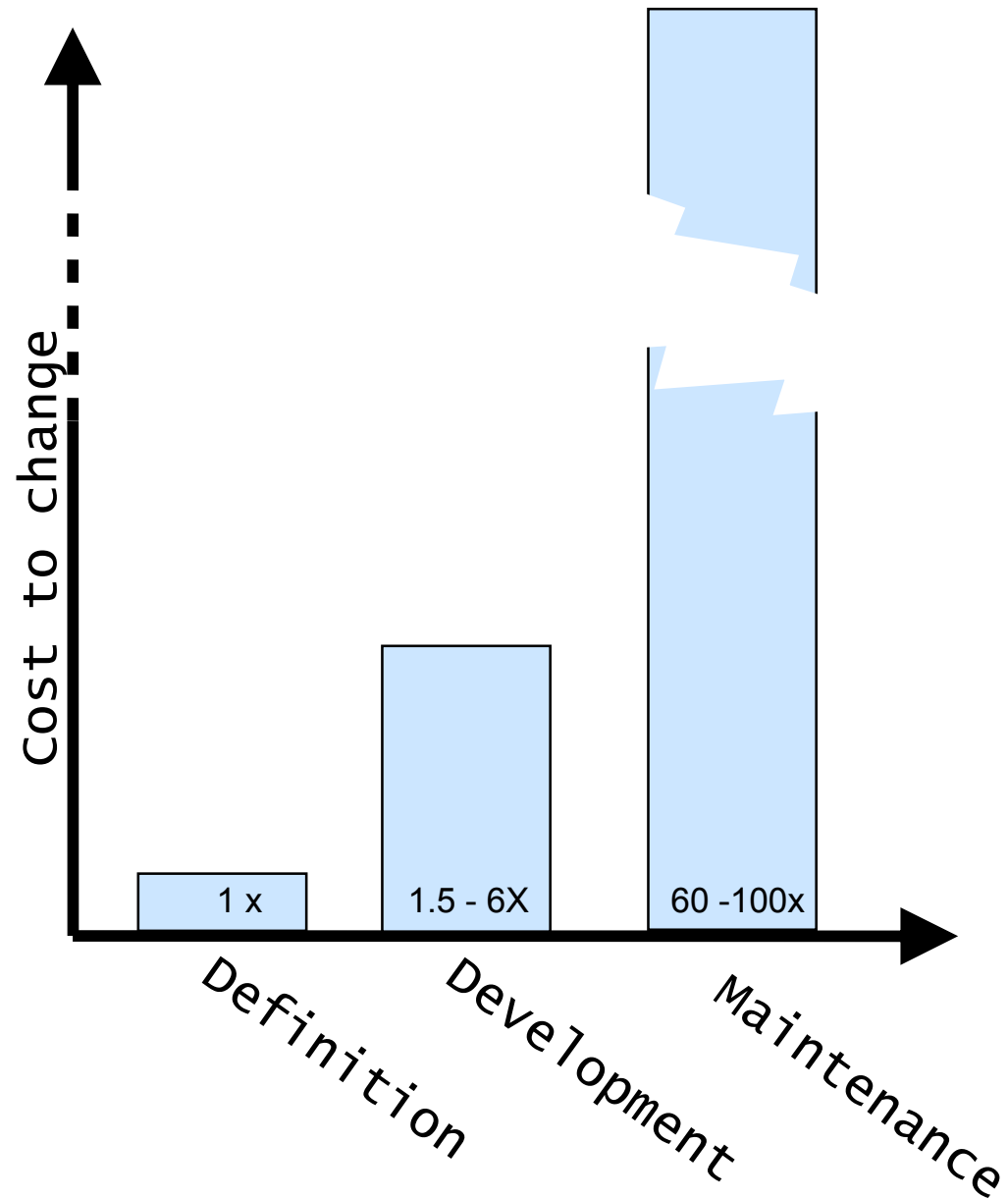
“Software Engineering”

“The establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines” (Pressman).

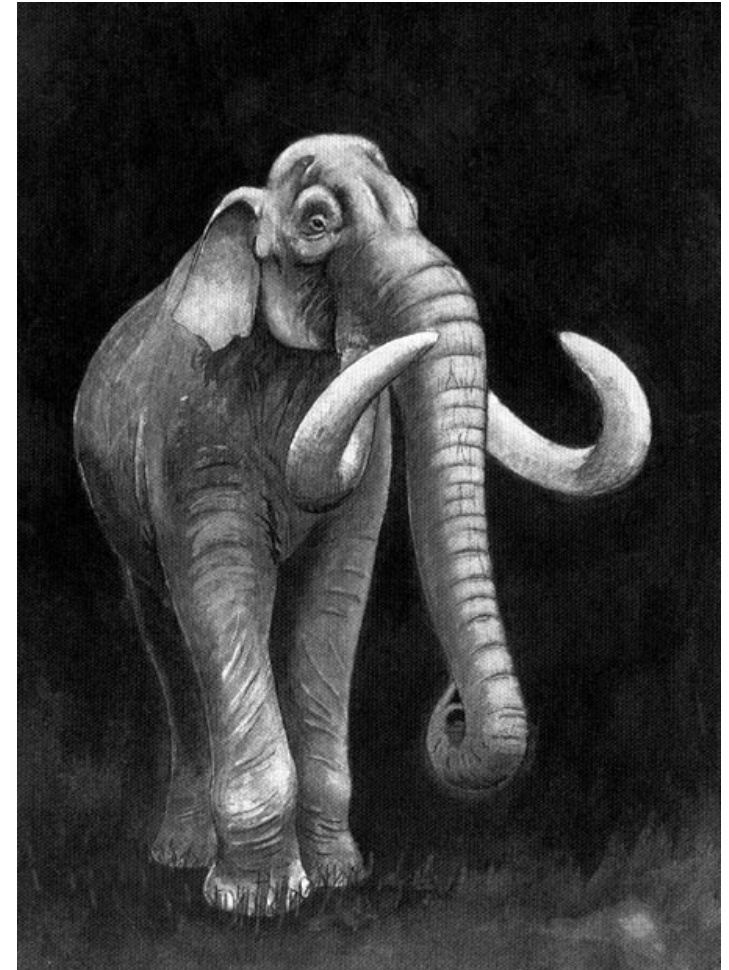
- Does this sound like any web project you’ve ever worked on?
- “Sound engineering principles” should mean calculability, predictability, and hence accurate costing. This rarely happens.

Costs of Correcting Software Errors

There's always some horror slide in any methodology pitch that looks like this.



- Enterprise A uses the Rational Unified Process.
- This proceeds at the speed of continental drift and has a lot of validation, iterations, meetings, and generates thousands of pages of documentation, and occasionally a working system, which will be written in Java.
- This will be delivered two years late but keep lots of J2EE engineers employed.



- Enterprise B uses Agile Development and Extreme Programming
- They have a standing meeting at 9am every morning
- Every two weeks they plan the next two weeks based on “user stories” (warm fuzzy term for use cases), admitting that they are incapable of estimation
- Project will never be finished, but bits of it will work well, and keep lots of Java programmers employed for a long long time.
- It should come as no surprise that XP was invented in the car industry.



Methodology 3

- Startup C is founded by two guys in a garage.
- They discuss what they want to build between building it.
- Usually some decent working code is produced.
- Funding is obtained, 100 programmers hired, and with no process or plan for managing 100 programmers, they buy a foosball table, VCs fire founders, and company either goes broke or finds a process that works.



Those are extremes.

- The trick to getting it right is:
 - Use only what works
 - SE tools are just that, tools
 - Use as much as is necessary and sufficient to get the job done.

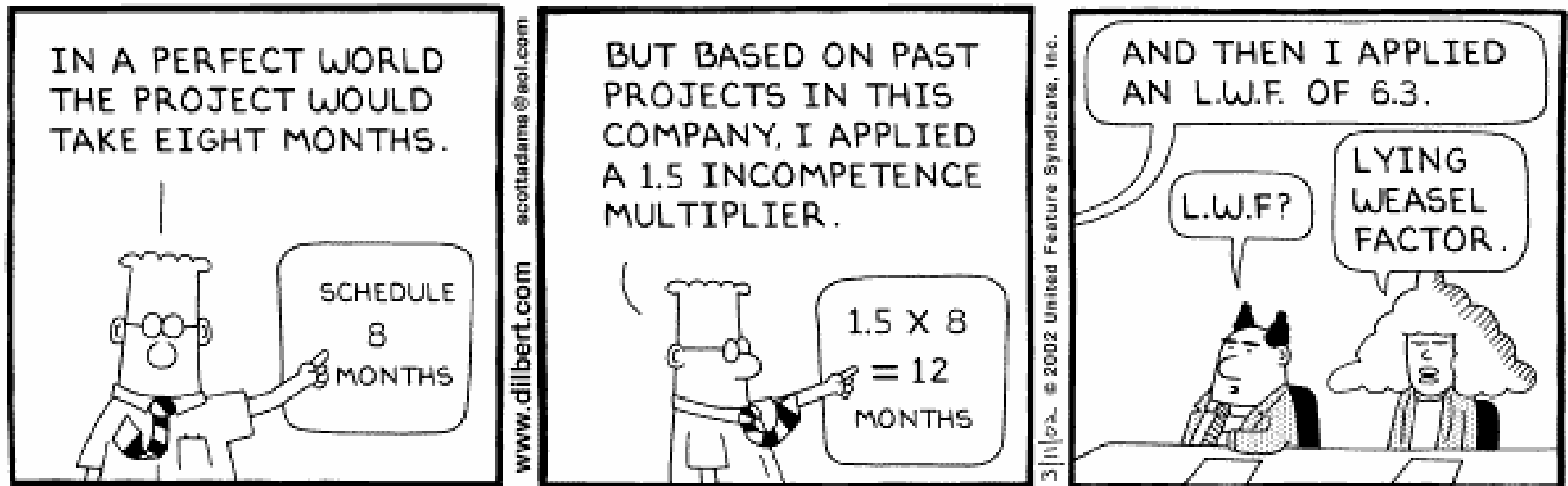
Why bother?

- Having some kind of process:
 - Helps developers (old, new, and junior) to understand what's going on
 - Helps the managers and clients understand what's going on
 - Helps to control scope, manage time, and manage cost

- The hardest part of the whole thing is estimation.
- Every one is bad at this, and the only way to get good at it is to practice. Always check how long it took versus your estimate, so your next estimate will be better.
- Half the problem is being asked to estimate too early. Before analysis it's impossible to do right. After analysis you have some idea. After design it's clearer.

How to estimate

- Understand enough of the problem and how complex it is.
- Break the solution down into small enough bits that you have a concrete idea of what needs doing. (Often difficult)
- Give that list to two other people. Each of you should estimate how long it would take.
- Take the longest estimate and multiply it by three.
- You will have forgotten Testing, QA, Documentation. Add those. Estimate. Multiply by three.
- Add this to the number you originally thought of, and add a bit.



Copyright © 2002 United Feature Syndicate, Inc.

- Use diagrams
- Keep it terse or no one will read it
- Make it easy to navigate and discuss:
Number pages, number items, have a table of contents
- Use version control
- Know the audience
- Know the purpose

Client feedback

- Every project has a client, internal or external
- Keep them up to date with what and how you are doing, and if you're going to miss deadlines
- Clients love to be proactively contacted and kept in the loop. Feeling involved helps to get them to give you more leeway when you inevitably miss deadlines.

Death march



</rant>

(Rant 2.0)

- If you know us, you might be surprised to see this section in here.
- PHP vs framework oriented languages (RoR)
- MVC and other TLAs
- What's happening in PHP frameworks: Zend, eZ, Cake, Symphony
- The application as framework

Frameworks and Architectures: use and abuse

- PHP frameworks are proliferating, and Rails doesn't help.
- Having an architecture like MVC can be a really good thing, but:
 - Everybody has a different idea about how this ought to be implemented
 - Some of the ideas are really twisted
 - Some make it hard to do very basic things simply
 - Code bloat
- Which framework?
 - No dominant paradigm yet, ergo little help with maintainability

Have a clear, simple, architecture that is easy to add to, easy to explain to new developers, and easy to remember now or in two or five years' time.

Shooting yourself in the foot

- C: You shoot yourself in the foot
- PHP: You shoot yourself in the foot with a gun made from pieces taken from 300 other guns
- Ruby on Rails: You want to shoot yourself in the foot, but the convention is to shoot yourself in the head.
- Frameworks are about *convention*
- If you know the framework, straightforward tasks become simpler
- Complex tasks are still going to be complex.

MVC

- Many (most) frameworks on offer are MVC focused.
- MVC is a design pattern, and as such everybody has a different idea how it ought to be implemented
- This means each framework is different. While familiarity with MVC gets you up to speed pretty quickly, your expectations as to the convention may be different. This can be counterproductive.

- There are plenty of component based frameworks (eZ components for example), and some of the MVC frameworks also have components (Zend)
- Trend for people using CMSes as their application framework...beginning with Wordpress (surprising numbers), Drupal, or Mambo, and hacking it into what they need.
- Some of the web service APIs can also be seen as an application framework, and I predict more work in that direction

- We'll run through a quick example of MVC using ZF
- Zend Framework just hit 1.0

- You'll need mod_rewrite installed
- You'll need:
 - Zend framework somewhere in your include path
 - .htaccess and index.php in your document root (note subdirs do not work at present without hacking on the Framework)
 - A directory structure somewhere *outside* your document root for your application

Zend framework

- Download from <http://framework.zend.com>
- Untar/zip it wherever you like
- Add this dir into your `include_path` in `php.ini`

```
application/  
  controllers/  
    IndexController.php  
  models/  
  views/  
    scripts/  
      index/  
        index.phtml  
    helpers/  
    filters/  
html/ ← DocumentRoot  
  .htaccess  
  index.php
```

- You'll need to set up mod_rewrite in your .htaccess file as follows (assuming Apache)

```
RewriteEngine on
```

```
RewriteRule !\.(js|ico|gif|jpg|png|css)$ index.php
```

- In `html/index.php`

```
<?php  
require_once 'Zend/Controller/Front.php';  
Zend_Controller_Front::run('/path/to/application/controllers');
```

- In application/controllers/IndexController.php

```
<?php
/** Zend_Controller_Action */
require_once 'Zend/Controller/Action.php';

class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
    }
}
```

- In application/views/scripts/index/index.phtml

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />
  <title>Index view</title>
</head>
<body>
  <p>Content of index page goes here.</p>
</body>
</html>
```

- Now at 1.0
- The MVC stuff has evolved through the versions and is now much improved...worth a revisit if you looked at something earlier.

- PHP has a large number of native database libraries
- In PHP 5, the MySQL lib has been improved (hence mysqli)
- There are also a number of database abstraction layers, including, but not limited to PEAR:DB and PDO

- Should I use a native library?
- Should I use a database abstraction layer?
- Which one should I use?
- Which one is better?

- mysql
- Database abstraction layers: general remarks
- PEAR:DB
- PDO
- Recommendations

What is mysqli?

- A new native driver for MySQL
- A PHP extension designed to work with MySQL 4.1 and above
- (The PHP MySQL library works with all versions of MySQL, but mysqli offers more functionality and an (optional) OO interface)

Why use mysqlcli?

- OO interface
- Uses new MySQL binary protocol from 4.1 (data sent to the server in binary format)
- Prepared statements
- Ability to set advanced connection options
- Better authentication

Example - procedural

```
$db = mysqli_connect('localhost', 'user',  
                    'pass', 'world');  
$query = "select name, district  
          from city  
          where countrycode='AUS'";  
$result = mysqli_query($db, $query);  
while ($city = mysqli_fetch_object($result))  
{  
    echo $city->name.', '.$city->district;  
    echo '<br />';  
}  
mysqli_free_result($result);  
mysqli_close($db);
```

```
$db = new mysqli('localhost', 'user',  
                'pass', 'world');  
$query = "select name, district  
          from city  
          where countrycode='AUS'";  
$result = $db->query($query);  
while ($city = $result->fetch_object())  
{  
    echo $city->name.', ' . $city->district;  
    echo '<br />';  
}  
$result->close();  
$db->close();
```

- OO function calls have the same names but with the `mysqli_` missing from the start
- Some procedural functions become properties e.g. `mysqli_num_rows()` vs `$result->num_rows`
(not a function)
- `mysqli` does not use studlyCaps unlike other PHP5 extensions

Example – prepared statements

```
$query = 'select name, district from city
          where countrycode=?';
if ($stmt = $db->prepare($query) )
{
    $countrycode = 'AUS';
    $stmt->bind_param("s", $countrycode);
    $stmt->execute();
    $stmt->bind_result($name, $district);
    while ($stmt->fetch())
    {
        echo $name.', '.$district;
        echo '<br />';
    }
    $stmt->close();
}
$db->close();
```

- Statement is prepared at DB server
- Can be executed multiple times with different data
- Saves time – query only sent once and planned once, execute multiple times
- Particularly useful for e.g. multiple inserts
- Data that will change is marked with question marks (?)
- Protects against SQL injection attacks

- Using a generic database library to connect to your database
- Previously many options existed:
 - phplib (PHP3)
 - PEAR::DB (PHP4)
 - Many other downloadable solutions, some good, some less so
 - And of course: homebrew
- Now: PDO (PHP Data Objects)

- For:
 - Always the same, whether using MySQL, Oracle, SQL Server, PostgreSQL, DB2, Firebird,...
 - Switch databases easily
 - Write code that will work with multiple databases (good for shared application development)
- Against:
 - Slower
 - Can't always use database specific features (lowest common denominator effect)
 - Emulation of features can be slow / confusing
 - How often do you really want to switch databases? Will changing the db calls in PHP really be your biggest problem?

- A classic database abstraction layer
- Supports fbsql, ibase, informix, msql, mssql, mysql, mysqli, oci8, odbc, pgsql, sqlite and sybase
- Written in PHP
- Features:
 - Portability modes (choose which features to emulate) e.g. **DB_PORTABILITY_NUMROWS**
 - OO or procedural interfaces
 - Prepared statements
 - Get at results in various formats

```
require_once 'DB.php';

$db =&
    DB::connect('mysqli://world:world@localhost/world');
if (PEAR::isError($db)) {
    die($db->getMessage());
}

$query = 'select name, district
        from city
        where countrycode=?';

$stmt = $db->prepare($query);
```

```
$result = $db->execute($stmt, 'AUS');

while ($city = $result->fetchRow(DB_FETCHMODE_OBJECT))
{
    echo $city->name.', '.$city->district;
    echo '<br />';
}

$result->free();
$db->disconnect();
```

What is PDO?

- PDO is not actually a database abstraction layer, but a data access abstraction layer.
- Use the same functions to query and fetch
- No SQL rewriting; no feature emulation (with one exception)
- If you try to use an unsupported feature you will get an exception
- Supports prepared statements, transaction control, control over error handling, scrollable cursors, support for LOBs, persistent connections, iterators...

- Written in C (not PHP) hence fast
- Allows use of database specific features – generic PDO driver and then a PDO driver for each specific database (think DBD/DBI)
- PDO drivers are used instead of native drivers, rather than as an extra layer on top.

- Install both the general PDO library and the specific one for the your database
- Unix: Download both with pear installer and follow usual install procedure
 - `extension=pdo.so`
 - `extension=pdo_XXX.so`
- Windows: Download newest set of PECL binaries, or just upgrade to 5.1
 - `extension=php_pdo.dll`
 - `extension=php_pdo_XXX.dll`
- Extensions must be loaded IN THAT ORDER

```
$dsn='mysql:host=localhost;dbname=world';  
try  
{  
    $db = new PDO($dsn, $user, $password);  
}  
catch (PDOException $e)  
{  
    echo 'Connect failed:'. $e->getMessage();  
}  
$query = "select name, district  
        from city  
        where countrycode='AUS'";
```

```
$stmt = $db->prepare($query);  
$stmt->execute();  
  
while ($city = $stmt->fetch(PDO::FETCH_OBJ))  
{  
    echo $city->name.', '. $city->district;  
    echo '<br />';  
}
```

- Supported databases: MySQL, PostgreSQL, Oracle, SQLite, Firebird, ODBC, MSSQL, Sybase, FreeTDS
- The only feature that is emulated where databases don't support it is prepared statements (hence providing SQL injection protection)

- For speed use native driver
- Note that PDO represents only a ~10% slowdown over native driver – in many cases the advantages outweigh the disadvantages
- For portability use some kind of abstraction: PDO gives a good compromise
- If you must have feature emulation use PEAR:DB
- For security use prepared statements – PDO gives portability with these
- For cool features use PDO

Section overview

- PHP4 vs PHP5 OO (How much do you know?)
- Upgrading issues

OO as best practice

- PHP developers still tend to err on the side of procedural code
- OO is one way to make your code modular
- PHP5 has “real” Java-like OO
- However...

- PHP5 is mostly backwards compatible
- There will be a limited number of PHP4 scripts that will not run on PHP5
- If you have your error reporting set high enough, you may see new warnings or notices
- **There are, however, serious gotchas.**

What will this code output?

```
class myClass
{
    var $var = 1;
}

$a = new myClass ();
$b = $a;
$b->var = 2;
echo $a->var;
```

- **Bonus question: How many instances of myClass will be constructed?**

- ... It depends
- PHP4: 1
- PHP5: 2

What if I knew that?

- If you want pass by value behavior, and want objects copied, use the PHP5 clone operator

- PHP4

```
$b = new b;
```

```
$c = $b;
```

- PHP5

```
$b = new b;
```

```
$c = clone $b;
```

What will this code output?

```
class a
{
    function a()
    {
        echo "Constructor of a called <br />\n";
    }
}
class b extends a
{

}
$b = new b;
```

- ... It depends
- PHP3: nothing
- PHP4/5: Constructor of a called
- In PHP4 or 5, a parent class' constructor will be called if a class does not have one
- In PHP5, you can deliberately call any function from the parent using the keyword parent and the scope resolution operator

```
parent::function() ;
```

What will this code output?

```
class b
{
    var $storage;
    function b() {
        $this->storage = new a();
    }
    function getStorage() {
        return $this->storage;
    }
}
$foo = new b();
echo $foo->getStorage()->myFunc();
```

- PHP5 allows this type of indirect reference

```
echo $foo->getStorage() ->myFunc();
```

- In PHP4 you will get a parse error and would need to rewrite it as:

```
$temp = $foo->getStorage();  
echo $temp->myFunc();
```


- Consider illegitimate uses of your application
- Educate yourself
- If nothing else, filter all external data

(From the PHP Security Guide at
<http://phpsec.org/projects/guide/>)

- External data is not to be trusted.
- What's external data?
 - Anything from a form
 - Anything from `$_GET`, `$_POST`, `$_REQUEST`
 - Cookies
 - Some server variables (e.g. `$_SERVER['SERVER_NAME']`)
 - Database query results
 - Web services data
 - Files
- The basic principle is to filter input and escape output
- Filter input using whitelisting where possible
- Escape output according to where it's going.

- Some common problems:
 - register_globals issues
 - XSS (Cross Site Scripting)
 - SQL/Command Injection
 - Code injection
- Other things to worry about:
 - Exposed source code
 - Session fixation
 - Session hijacking
 - Cross site request forgeries (CSRF)
 - Cross domain Ajax requests

Attack: register_globals

- At PHP 4.2, the register_globals setting in php.ini was switched from On to Off by default.
- Broke everybody's code
- Raised the barrier to entry for newbies:
\$_REQUEST['myvar'] vs \$myvar
- Why?

```
<?php
```

```
if ($username == 'admin' &&  
    $password == 'secret')
```

```
{  
    $valid_user = true;  
}
```

```
if ($valid_user)  
{  
    echo "Logged in as admin";  
}
```

```
?>
```

Settings aren't the issue

- The issue is about uninitialized variables, rather than really being about `register_globals`.
- Even with it off, programmers can and do run `extract($_REQUEST)`. Hardened-PHP turns this off, but programmers can of course write their own version.
- Sometimes you will need `register_globals/extract` for backwards compatibility (until you upgrade your codebase, not necessarily a trivial task)
- It's a developer education issue (as are a lot of these).
- You can help developers to get educated by turning up the `error_reporting` level. (More on this later)

- An attack by a malicious user where they enter some data to your web application that includes a client side script (generally JavaScript).
- If you output this data to a web page without filtering it, this script will be executed.

```
<?php
if (file_exists('comments'))
{
    $comments = file_get_contents('comments');
}
else
{
    $comments = '';
}
if (isset($_POST['comment']))
{
    $comments .= '<br />' . $_POST['comment'];
    file_put_contents('comments', $comments);
}
?>
```

XSS Example – part 2

```
<form action='xss.php' method='POST'>  
  Enter your comments here: <br />  
  <textarea name='comment'></textarea> <br />  
  <input type='submit' value='Post comment' />  
</form><hr /><br />  
  
<?php echo $comments; ?>
```

So what?

- So it's JavaScript (or even plain old HTML), I hear you saying, so what? It runs on the browser.
- What can I do with that?
- Heaps of badness:
 - Annoying popups
 - Meta-refresh
 - DOM rewriting
 - Dubious forms
 - Steal cookies (which can then set up a session attack)
 - AJAX (XMLHttpRequest)

How do I prevent this?

- Filter output to the browser through `htmlentities()`.

More Realistic Examples

```
echo "Top 10 search results for $input";
```

```
echo "Sorry, $input is not a valid phone  
number";
```

```
echo "<input type='text' value='$input'>";
```

```
echo "logged in as $username";
```

- Enter SQL in e.g. form fields in such a way that it is executed on the web app database.
- A variation is command injection, where user data is passed through `system()` or `exec()`.
- It's basically the same attack.
- (Code injection is also a variation, but we'll talk about that separately)

SQL Injection Example

```
$username = $_POST['username'];  
$password = $_POST['password'];  
$query = "select * from auth where username = '" .  
    $username . "' and password = sha1('" . $password . "')";  
$db = new mysqli('localhost', 'demo',  
    'secret', 'security_demo');  
$result = $db->query($query);  
if ($result && $result->num_rows)  
{  
    echo "<br />Logged in successfully";  
}  
else  
{  
    echo "<br />Login failed";  
}
```

- Most databases have a native escape command
- For example, with MySQL you should use `mysql_real_escape_string()` (similar to `mysql_escape_string()` which is now deprecated, takes a character set as an optional argument)
- Don't rely on magic quotes.
- Best solution: use prepared statements

Just Say No To Magic Quotes



Attack: Code injection

- While this can be grouped with SQL injection and command injection, it's a serious enough and common enough problem to merit its own slide
- Problem occurs when you accidentally execute arbitrary code, typically via file inclusion
- Poorly written code can allow a remote file to be included and executed as though it were a trusted local file
- Remember that many PHP functions such as `require` can take an URL or a filename.
- Passing user input as a filename or part of a filename invites users to start filenames with `http ...`

Code Injection Example

Theme Selector

```
<form>Choose Theme:  
<select name = theme>  
<option value = blue>Blue</option>  
<option value = green>Green</option>  
<option value = red>Red</option>  
</select>  
<input type = submit>  
</form>  
<?php  
    if($theme) {  
        require($theme.'.txt');  
    }  
?>
```

- In many cases, whitelist
- Filter user input
- Disable *allow_url_fopen* setting in php.ini. This disables require/include/fopen of remote files.

- Some basic principles (again):
 - Don't rely on server configuration to protect you (e.g. magic quotes)
 - Design your application with security from the ground up: eg, use a single line of execution that begins with a single point of data cleaning.
 - Review your colleagues' code and then yours
 - Seek advice from experts (security audit)
 - Educate yourself and your developers and where possible make it easy for your staff to do the right thing.
 - Keep your install up to date. Stay on top of patches and advisories.

- Make the first step in any script that accepts input to clean the data
- Some advocate e.g.

```
$clean[ 'foo' ] = intval($foo) ;
```

Dispatch architectures

- A dispatch architecture (single entry and exit point) is a popular solution to making sure you have cleaned your data
- MVC works as well, with slightly more discipline required

Using error reporting

- Error reporting is for you, not for end users
- [Error reporting](#) slide

- Many problems arise because developers are not educated
- Code review from peer to peer, and importantly, from more experienced to less experienced staff can really help
- As well as formal code review processes, take the time to mentor junior staff (and read those commits)
- Developer education is often the most valuable part of a security audit

- This is the process where code is analyzed, both to look for specific vulnerabilities, and for general recommendations
- Can be done by external consultants: this avoids the refactoring problem (only gets done when somebody has the time to do it on an ad hoc basis). Consultants should be up to date.
- Should be accompanied by developer training to learn how to fix the problems, and to make sure the same problems don't arise again.

- One of the most important aspects
- If you are not responsive to:
 - Reported exploits and bulletins
 - Code patches
 - Feedback about how to improve your codethings will never improve.

Performance Tuning

- Basic guidelines: planning for growth
- Profiling to find bottlenecks
- Common problems
- Caching

- Write your PHP code efficiently:
 - Minimize LOC to be parsed
 - Minimize files opened
 - Minimize database queries
 - Minimize connections to outside machines
 - Develop with caching in mind
 - Profile and Benchmark
- Write your app so that you can add load balancing or replication later (caution is needed with sessions)

- 1 web server: sessions on disk
- Multiple servers with load balancing == multiple solutions:
 - Sticky sessions (defeat the purpose)
 - Session database
 - Client side session data (but remember security implications)

Using Environment Variables

- You will probably end up with more than one web server
- There will probably be deliberate variations between them
- Setting environment variables can be a good way to
 - Store DB passwords securely on shared machines
 - Detect if this is test or live server
 - Detect which version of the template/language/code to use
 - Track paths to directories that vary on machines

Example

```
$db = mysqli_connect($_SERVER['DB_HOST'],  
                    $_SERVER['DB_USER'],  
                    $_SERVER['DB_PASSWORD'],  
                    $_SERVER['DB_NAME'] );
```

- When looking for performance problems there are several important rules:
 - Optimization is almost always premature
 - Never make assumptions about which parts of your application are slow
 - Use an empirical tool such as APD or xdebug
- Having said that of course, there are some guidelines as to what is generally slow

- Xdebug is a profiling and debugging tool
- I'm using this today rather than APD because I'm on Windows, but APD is also very good
- Get xdebug from <http://xdebug.org>
- You will also need one of Kcachegrind/Callgrind/Wincachegrind

- *nix: `./configure --enable-xdebug`
- Windows:
- `zend_extension_ts="c:/php/ext/php_xdebug.dll"`
- Then add the following to your `php.ini`:
- `xdebug.profiler_enable = 1`
- `xdebug.profiler_output_dir = /tmp/xdebug`

- While you have xdebug enabled, each script execution will dump a cachegrind file in the directory you specified.
- Open this in whichever version you installed.

Sample output

WinCacheGrind - [C:\Program Files\Apache Group\Apache2\htdocs\pmsec\sql_injection.php (cachegrind.out.3667002263)]

File View Profiler Tools Window Help

sql_injection.php

sql_injection.php

{main}

C:\Program Files\Apache Group\Apache2\htdocs\pmsec\sql_injection.php

File: C:\Program Files\Apache Group\Apache2\htdocs\pmsec\sql_injection.php

Self time: - (-) Cumulative time: 433ms (100.00%)

Line by Line Overall

Find: Regular expression

Function	Avg. Self	Avg. Cum.	Total Self	Total Cum.	Calls
{main}	0.2ms	433ms	0.2ms	433ms	1
php::mysqli->query	397ms	397ms	397ms	397ms	1
php::mysqli->mysqli	36ms	36ms	36ms	36ms	1

Sum of total self time: 433ms (100.00%) Sum of calls: 3

Num.	Self	Cum.	Called by	Called from	Stack trace
1	0.2ms	433ms	sql_injection.php	sql_injection.php	

File Name | Title | Modified | Size

Allocated memory: 113,380 bytes

- Typical causes of poor performance include:
 - Poorly formed DB queries, or executing too many queries on each page (70-80 and up)
 - Overuse of includes/requires
 - Splitting content across too many files
 - Including huge libraries to use a single function
 - Poorly formed regexes
 - Dependence on external resources (e.g. web services APIs)
 - HTML/CSS/Image bloat

- For CMSes etc where the output is basically static, cache the HTML
- For data that's expensive to retrieve or calculate, cache the results of the calculation (sqlite is useful here)
- Use a reverse proxy cache
- Use your database's query cache

Storage engines

What are storage engines?

- These are sometimes called storage engines and sometimes called table types
- MySQL supports several, and the number is growing. We'll look at:
 - MyISAM
 - InnoDB
 - MEMORY (nee HEAP)
 - MERGE
 - Also BDB, CLUSTER, ARCHIVE, BLACKHOLE, CSV, Falcon, community engines, etc, etc, etc
- Each table in a database can be of a different type

Why use different storage engines?

- Each storage engine has its own capabilities and limitations
- The default type is MyISAM so this is what most web developers use
- Solve particular problems with different table types
- Use different table types to access different functionality

- Default type
- Fastest for many applications
- Full text searching
- Easy to check and repair
- Can be compressed

- No transactions
- No foreign keys
- Source of many gotchas

- Transaction safe table type
- Supports transactions and foreign keys
- Row level locking
- Tables can be of any size
- Used successfully by many large web sites
- Faster than MyISAM in situations where you have many reads and writes happening at the same time (eg busy web forum)

- Tables stored completely in memory
- Use hashed indexes
- If power lost data will be lost
- Some limitations:
 - Should limit max number of rows to avoid eating memory
 - No BLOB/TEXT/AUTO_INCREMENT
- Useful for temporary tables
- Previously called HEAP tables

- Collection of MyISAM tables that can be treated as one table
- Useful for working around max filesize operating system limitations when you want to use MyISAM

How is this useful?

- The two principal types you will use as a developer are MyISAM and InnoDB
- Remember you can use a mix of table types within the same database:
 - MyISAM when you are doing many SELECTS or INSERTS (but not both) on one table for max speed
 - e.g. product info or news article retrieval
 - InnoDB when transactions are important – e.g. financials
 - MEMORY for static data
 - MERGE for very large MyISAM tables (good for logs)

- You can change storage engines with an ALTER TABLE statement if you get it wrong (with obvious limitations)
- `CREATE TABLE blah () TYPE=INNODB;`

Query caching

What is query caching?

- Available from 4.0.1
- Each SELECT query and the result are stored in the cache
- The query cache can give the same answer again very quickly for a SELECT query that has been run recently
- This adds an overhead for distinct queries, so it is disabled by default

Advantages

- For web applications, with queries coming from scripts, the same SELECT queries are likely to repeat
- This is particularly true for
 - Page templating systems
 - Catalogues
 - Discussion forums
 - Blogs
- Turn it on!

- Any change to a table flushes the query cache of entries relating to that table.
- Output of prepared statements is not cached.

- Tip: Using PDO and emulating prepares allows you to use the query cache.

- By default, the directive `query_cache_size` is set to 0
- In `my.cnf` or `my.ini` enable by giving it a size

```
query_cache_size=32M
```


- Configure MySQL securely
- Understand the privilege system, and use it appropriately
- Use encryption when needed
- Don't trust user data (more on this later)

- Simple principles:
 - Don't run `mysqld` as (Unix) *root*. Run it as a user created specifically for this purpose, e.g. *mysql*. Don't use this account for anything else. (Note that the MySQL *root* user has nothing to do with Unix users so this doesn't affect MySQL internally at all.)
 - Set permissions on the database directories so that only your `mysqld` user (e.g. *mysql*) can access them.
 - Disable symlinks to tables with *--skip-symbolic-links*.
 - Disallow access to port 3306 (or whatever port you have MySQL running on) except from trusted hosts

Accounts and Privileges

- All MySQL accounts need a password, especially *root*. (Don't forget anonymous users, either.)
- Grant users the minimum level of privilege required to do their job. (Principle of Least Privilege)
- Some privileges require special attention:
 - Only the root user should have access to the mysql database, which contains privilege information
 - Keep FILE, PROCESS, and SUPER for administrative users. FILE enables file creation, PROCESS allows you to see executing processes (including passwords in plaintext), and SUPER can be allowed to e.g. terminate client connections.
- Avoid wildcards in hostnames in the host table.
- Use IPs instead of hostnames in the host table if you don't trust your DNS

- Don't store application passwords in plaintext in the database. (Use one way hashing)
- Require remote database connections to be via ssh or tunneled through it
- Avoid old MySQL passwords (pre 4.1). (Disable with *--secure-auth*, and avoid use of *--old-passwords*.)

Performance tuning

- Performance tuning basics
- Replication gets its own section

- The same goes as for PHP tuning: what you think is slow may not be slow.
- Benchmarking is the only way to truly test this.
- When tuning, change one thing at a time
- Your toolkit:
 - EXPLAIN
 - Slow Query Log
 - mytop
 - More, better tools all the time

- The most common problem that comes up on MySQL mailing lists is slow query performance due to lack of indexing
- Create relevant indexes. Make sure your queries use them. EXPLAIN is your friend here.
- The order of multi-column indexes is important
- Remove unused indexes to speed writes

- Use the smallest data type possible
- Use fixed width rows where possible (prefer char over varchar: disk is cheap)
- Denormalize where necessary (see next slide)
- Take static data out of the database or use MEMORY tables
- Use the appropriate storage engine for each table

- Minimizing the number of queries is always a good start. Web pages that need to make 70-80 queries to be rendered need a different strategy:
 - Cache the output
 - Cache part of the output
 - Redesign your schema
 - Decide if you can live without some queries.
- Confirm that your queries are using the indexes you think that they are
- Avoid correlated subqueries where possible
- Stored procedures are notably faster

- Replication can be a solution to scaling up your database
- MySQL only really supports master-slave configurations, in which writes are made on a single master, and are then replicated through to one or many slaves
- This is suitable for environments in which writes represent roughly 10% or less of queries

- Set up an account for the slave on the master, with REPLICATION SLAVE privileges.
- Login via the monitor, and execute
FLUSH TABLES WITH READ LOCK;
- Leave that connection open (so the tables stay locked)
- Open a shell and tar up the data directory for the database you want to replicate, typically /usr/local/mysql/data/[dbname]

- In your monitor window type **SHOW MASTER STATUS;**

- The output will be similar to:

```
+-----+-----+-----+-----+
| File           | Position     | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| dbname-bin.000715 | 196626184   |               |                   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

- Note down the data under “File” and “Position”.
- Then:
UNLOCK TABLES;

- Shutdown MySQL if it's running
`mysqladmin -u root shutdown`
- Copy across the tar file from the master, and untar it in the data directory on the slave.
- Restart slave with the skip-slave-start option:

```
/usr/local/mysql/libexec/mysqld --skip-slave-start
```

- Open a mysql client on the slave
CHANGE MASTER TO
MASTER_HOST='host',
MASTER_USER='replication_user_name',
MASTER_PASSWORD='replication_password',
MASTER_LOG_FILE='recorded_log_file_name',
MASTER_LOG_POS=recorded_log_position;
START SLAVE;

- On the slave, you should see two threads running: an I/O thread, that reads data from the master, and an SQL thread, that updates the replicated tables.
- (You can see these with SHOW PROCESSLIST)
- Since updates on the master occur in *multiple* threads, and on the slave in a *single* thread, the updates on the slave take longer.
- Slaves have to use a single SQL thread to make sure queries are executed in the same order as on the master
- The more writes you do, the more likely the slaves are to get behind, and the further behind they will get.
- At a certain point the only solution is to stop the slave and re-image from the master.
- Or use a different solution: multi master (not MySQL), data partitioning, etcetera.

Questions?




- PHP/FI – Huh?
- PHP/FI 2.0 – Huh?
- PHP 3.0 – OO features added, many missing. Essentially syntactic sugar
- PHP 4.0 – More OO features, but important ones missing. Backwards compatibility maintained
- PHP 5.0 – Complete OO featureset. Mostly backwards compatible, but many deprecated features.

OO concepts

customer
\$name \$address \$orderHistory
register() placeOrder() lookupOrder()

Class 

<u>customer:john</u>
\$name="John Smith" \$address="1 Foo St Bartown" \$orderHistory=array
register() placeOrder() lookupOrder()

Object 

```
class customer
{
    var $name;
    var $address;
    var $orderHistory;

    function register()
    {
        // some code here
    }
    ...
}
```

- To create a customer, we call **new** like this:
`$fred = new customer();`
- This creates an **object** named `$fred` that belongs to the **class** of objects named **customer**

```
function __construct($name)
{
    $this->name = $name;
    $this->address = '';
    $this->orderHistory = array();
}
```

- `__construct` is new in PHP5. For older versions, it must have the same name as the class

```
function __destruct()  
{  
    // do any cleanup  
    fclose($handle);  
}
```

- Destructors are new in PHP5.

```
$fred = new customer('Fred Smith');  
echo $fred->name;
```

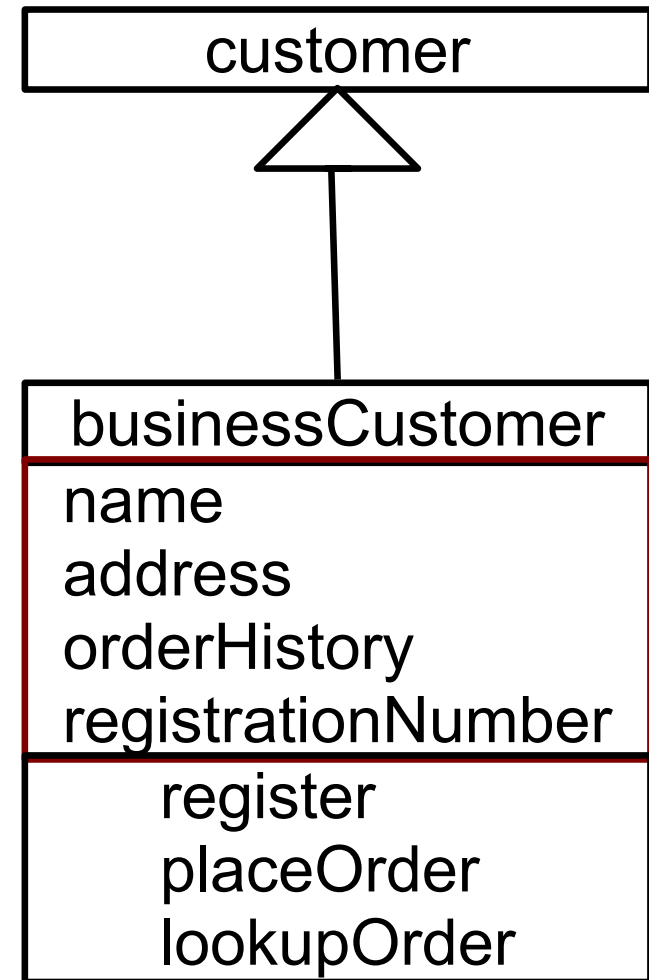
```
function __get($attribute)
{
    return $this->$attribute;
}
function __set($attribute, $value)
{
    if($attribute == 'name')
    {
        if($value == '')
            return;
    }
    $this->$attribute = $value;
}
```

```
class customer
{
    protected $name;
    protected $address;
    protected $orderHistory;
    public function register()
    {
        // some code here
    }
    ...
}
```

```
$fred = new customer();  
$fred->register();
```

OO Concept - Inheritance

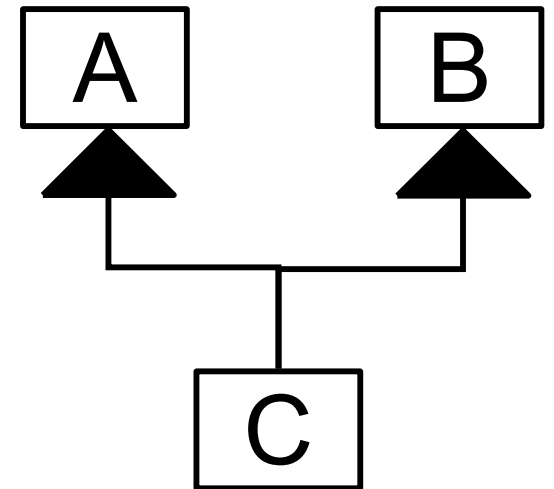
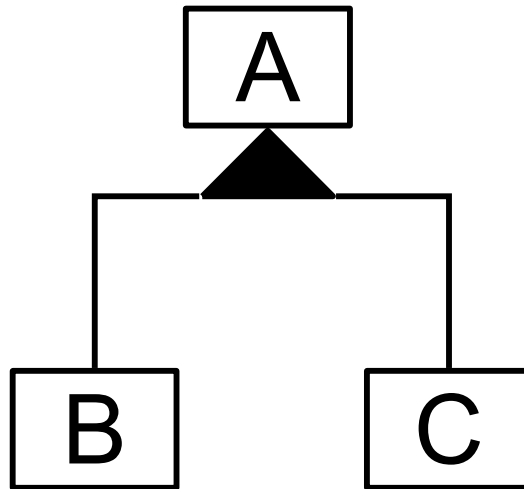
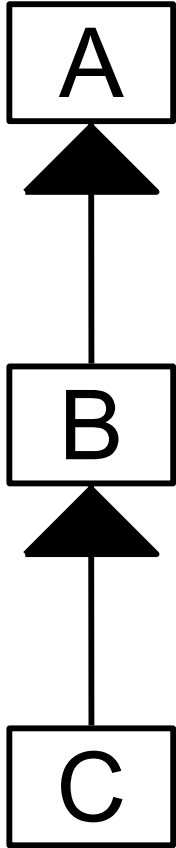
- One of the key benefits of OO software is that in many domains, you can efficiently deal with variation by treating some classes as variations of another



```
class businessCustomer extends customer
{
    protected $registrationNumber;

    public function register()
    {
        customer::register();
        // some code here requesting number
    }
    ...
}
```

No Multiple Inheritance



- Like Java, PHP5 provides interfaces to overcome most of the limitations of not allowing multiple inheritance
- An interface is essentially a promise to implement a specific API

```
interface displayable
```

```
{
```

```
    public function displayAsHTML();
```

```
}
```

```
class businessCustomer extends customer  
    implements displayable
```

```
{
```

```
    function displayAsHTML()
```

```
{
```

```
    ...
```

- The main reason to use interfaces is to take advantage of polymorphism
- Polymorphism allows you to treat a mixed set of variables as though they are the same type.
- If we know that all classes that inherit from a particular class, or implement a particular interface will have a method, then we can call it on mixed types and let the system make sure that the right version is called

Polymorphism Example

```
$array = array();  
$array[] = new customer();  
$array[] = new businessCustomer();  
$array[] = new customer();  
  
foreach($array as $customer)  
{  
    $customer->register();  
}
```

- Anytime a method is called that does not exist, PHP will call `__call` to handle the situation.
- You can use this for overloading, or any other logic (the Zend framework uses it extensively, for example, in the MVC classes)
- Your code can decide at runtime what to do

```
class dynamicClass
{
    function __call($name, $args)
    {
        echo "You called $name, with ".
            count($args) . " parameters<br
        />\n";
    }
}
$test = new dynamicClass();
$test->myFunc();
$test->myOtherFunc(20);
```

```
abstract class AbstractClass
{
    abstract public function foo();

    public function bar() {
        Echo "doing something";
    }
}
```

- PHP is a weakly typed language, but in PHP5 you can force object parameters to be a specific type

```
function MyFunction (MyClass $foo)  
{  
    echo "doing something";  
}
```


What are transactions?

- Transaction = set of SQL statements that can be guaranteed to either all be executed or all not be executed, leaving the database in a consistent state
- ACID =
 - Atomicity (treat statements in ts as a unit)
 - Consistency (DB will be before or after ts)
 - Isolation (uncommitted ts do not affect others)
 - Durability (committed ts must be persistent)
- Supported in InnoDB and BDB storage engine types, not by MyISAM

```
start transaction;
```

```
insert into t1 values (2, 2);
```

```
commit;
```

- Until the commit statement is executed, the inserted row is only visible within this connection (ie not to others)
- We can use `rollback;` to undo any SQL statements since the transaction started.

How is this useful on the web?

- Depends what you are doing
- For many web apps data integrity is not that important (controversial?)
- Use TST when:
 - Financial or other mission critical data is involved (often quantitative)
 - Data integrity is more important than pure speed
- You can have different table types in your database and therefore can apply transactions only to the parts of your application that need it

What are foreign keys?

- Integrity constraints
- Column c1 in table t1 must contain a value from column c2 in table t2
- Implemented in InnoDB

```
create table order_items
(
  orderid int unsigned not null references
  orders(orderid),
  isbn char(13) not null,
  quantity tinyint unsigned,
  primary key (orderid, isbn)
) type=InnoDB;
```


What are subqueries?

- A subquery is essentially a SELECT statement used inside another SQL query (also called a subselect)
- They are often interchangeable with
 - A query using a join
 - A script/program in another language that does two or more queries
- The subquery version of a query used to run slower than the equivalent join version, but can be very readable and maintainable.
- May now run much faster than the join version – benchmark for your own queries

- The population of Melbourne:

```
select Population
```

```
from City
```

```
where Name='Melbourne';
```

- Cities with a population greater than Melbourne:

```
select Name, Population
```

```
from City
```

```
where Population > 2865329;
```

- A subquery allows you to treat the result of a query as a value in other queries

```
select Name, Population
from City
where Population >
    (select Population
     from City
     where name='Melbourne' );
```

Table Subquery example

- A subquery does not have to return a single value and may itself contain subqueries

```
select distinct Name
from Country
where Code in (select CountryCode
               from City
               where Population >
                  (select Population
                   from City
                   where name='Melbourne' )
               );
```

Doing it with a script instead

- We could run 3 separate queries and use PHP (or whatever) to generate each query based on the data from the previous one:

```
select Population from City where name='Melbourne';
```

```
select distinct CountryCode from City  
where Population > 2865329;
```

```
select Name from Country where Code in  
( 'ARG' , 'AUS' , 'BGD' , 'BRA' , 'GBR' , 'CHL' , 'EGY' ,  
  'ESP' , 'IDN' , 'IND' , 'IRQ' , 'IRN' , 'JPN' , 'CHN' ,  
  'COL' , 'COD' , 'KOR' , 'MAR' , 'MEX' , 'MMR' , 'PAK' ,  
  'PER' , 'DEU' , 'SAU' , 'SGP' , 'THA' , 'TUR' , 'RUS' ,  
  'VNM' , 'USA' );
```

Doing it with joins instead

```
select distinct Country.Name
from Country, City City1, City City2
where
    City1.Population > City2.Population and
    Country.Code = City1.CountryCode and
    City2.Name='Melbourne' ;
```

Stored Procedures and Cursors

What are stored procedures?

- Long familiar to ORACLE programmers
- Relatively new to MySQL
- Functions that are stored inside the database
- Written in SQL + control structures

What are Cursors?

- A pointer into a result set
- From a PHP perspective, similar to a result resource (that you then access with repeated calls to e.g. `mysqli_fetch_result`)
- Only usable with stored procedures at present

```
# Basic stored procedure example
delimiter //

create procedure total_orders (out total float)
BEGIN
  select sum(amount) into total from orders;
END
//

delimiter ;
```

```
# Basic syntax to create a function
```

```
delimiter //
```

```
create function add_tax (price float) returns float  
return price*1.1;  
//
```

```
delimiter ;
```

- A function returns a value
- Procedures can return values via parameters
- Procedure parameters must be specified as IN or OUT or INOUT
- Function parameters are all IN by default

```
create procedure largest_order(out largest_id int)
begin
  declare this_id int;
  declare this_amount float;
  declare l_amount float default 0.0;
  declare l_id int;

  declare done int default 0;
  declare continue handler for sqlstate '02000'
      set done = 1;
  declare c1 cursor for select orderid, amount
      from orders;
```

```
open c1;
  repeat
    fetch c1 into this_id, this_amount;
    if not done then
      if this_amount > l_amount then
        set l_amount=this_amount;
        set l_id=this_id;
      end if;
    end if;
  until done end repeat;
close c1;
set largest_id=l_id;
end
```

- Kind of like an exception or interrupt
- `sqlstate 02000` = no more rows
- The continue handler is called when there are no more rows to fetch
- It sets a flag to terminate execution

- Like a result set in PHP, a cursor is associated with a query
- Each fetch pulls the next row from the query

- Note that you need a temporary variable to store the result
- Not placed in the OUT parameter until execution is finished

- If you plan to build interfaces in more than one language (say PHP for your Web app, and something else for the desktop version) you can have the same underlying functional code
- You can use stored procedures almost like an OO interface: that is to encapsulate data and hence control the way data is accessed or changed.

Full text searching

- Reasons to want to search text include
 - a site search
 - search a catalogue of products or articles
- Full text indexing and searching was added in 1999
- Boolean features added later
- Faster in newer versions
- Can be used on all text based fields

- There are various ways to search text
 - Within MySQL
 - LIKE for simple pattern matching
 - REGEXP for more complex patterns
 - Fulltext indexing and searching
 - Outside MySQL
 - Specific indexing and searching tools
 - ht://Dig - <http://www.htdig.org/>
 - mnoGoSearch - <http://search.mnogo.ru/>
- Indexing is the important part - LIKE and REGEXP are not useful for large DBs

Example

Search

All Products 



```
<?php
$db = new mysqli('localhost', 'user', 'pass',
    'phpcon');
// live dangerously, insert user input
// directly into a query
$searchTerm = $_REQUEST['searchTerm'];

$query = "select name, description,
    from products
    where
    match (name, description)
        against ('$searchTerm')";

$result = $db->query($query);
```

```
while($product $result->fetch_object())
{
    echo '<h1>'.
        $product->name.
        '</h1>'.
        '<p>Relevance: '.
        number_format($product->score,1).
        '</p>'.
        $product->description.
        '<hr />';
}
$result->close();
$db->close();
?>
```

- The query is a phrase in natural language
- Attempts to find rows that are ‘about’ the concept in the phrase

```
select name, description
from products
where match (name, description)
against ('blue widget')
```

- The query is an expression that may contain Boolean operators
- Find only rows that exactly match the expression

```
select name, description  
from products  
where match (name, description)  
against ('+blue -green'  
in boolean mode)
```

- + results must include this word
- - results must not include this word
- < > increase or decrease relevance of results that include this word
- ~ negative contribution to relevance
- * wildcard
- “...” phrase matching

- You need to understand how stop words work by default and change the default behaviour if it will not suit your application
 - A list of very common words are not indexed
 - Any word that appears in 50% of documents
 - Any word with 3 characters or fewer
- Not many people understand how to formulate a Boolean query
- Natural language queries run faster
- Only natural language queries return a relevance score