Database-enabled web technology (LIX021B05)

Instructor: Çağrı Çöltekin
c.coltekin@rug.nl

Information science/Informatiekunde

Fall 2011/12

Who, where, when

Course	DB-enabled web technology (LIX021B05) 2011/12
Instructor	Çağrı Çöltekin
Email	c.coltekin@rug.nl
Lectures	Wed 13:00–15:00, `1315.0031`
Labs	Tue 13:00–15:00, `1312.0107A C`
Office hours	Tue 15:00–17:00, `H1311.0426` (or by appointment)
Course page	`http://www.let.rug.nl/coltekin/courses/DBweb2011/`

Literature

There is no compulsory textbook for this course.

- Online resources will be provided for each subject.
- A good database book, such as one of the following will be handy to have at hand.
  - *Database System Concepts* by A. Silberschatz, H. F. Korth & S. Sudarshan.
  - *Database Management Systems* by Ramakrishnan & Gehrke
  - *A First Course in Database Systems* by Ullman & Widom
  - *Fundamentals of Database Systems* by Elmasri & Navathe
  - *Database Systems: The Complete Book* by Garcia-Molina, Ullman & Widom
  - *Database Systems: A Practical Approach to Design, Implementation, and Management* by Connolly & Begg

What you should already know

Databases You are assumed know basic relational database design concepts and SQL. We will only have a quick review/reminder session.

HTML You are assumed know the basics of HTML. We will only revisit/review HTML forms in this course.

Programming You are assumed to have some experience with programming. We will do a fast introduction to PHP, but you should be able to pick it up quickly.

About this course

This is an *introductory* course on web programming using databases. The particular focus will be on good database design (which you already know how to do) and secure web programming.

- The knowledge of relational database management systems, SQL, and programming skills are assumed.
- The course has a practical focus: you will develop a real-world web application using a database backend during the course.
- We will use PHP and MySQL, however skills you develop should be easily transferable to any programming language and database management system.

After this course . . .

You should be able to

- develop interactive server-side web applications with a relational database back end
- develop a relational database for a real-world application
- understand what happens behind the browser in a web-based application
- be able to identify potential security problems in web-based applications
- be familiar with performance issues for large scale web-based applications using databases.

Time plan

Week	Lecture (Wed)	Reading
1	Introduction & A crash course on PHP	none
2	DB design: an introduction/review	Previous course notes
3	Web programming: the background	TBD
4	SQL and programming & Accessing databases from PHP	TBD
5	Session Management & Security	TBD
6	Security (2)	TBD
7	High performance web applications & wrapping up	none

Evaluation

- Project: 60%.
  - Initial design report (Due date: Nov 28): 20%.
  - Final project/report: 40%.
- Written exam: 40%.

You need a minimum of 55% from each to pass.

About projects

Project form the main part of this course.

- Projects can (preferably) be done by a team up to 4 people (depending on the complexity of the project).
- Contribution by all members are compulsory.
- Naturally, you will share tasks, however, everybody should understand the project they are working on fully.
- You are required to use a version management system for all project related files (more on this today).
- You can pick the project yourself, however, a set of recommendations (with possible real-world use) are provided.

Time is short: you need to act quickly to team up, and decide for a project.

Project suggestions

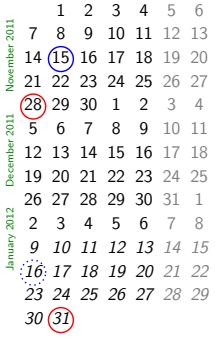
Some projects with customers:

- A collaborative word-net creation/management system. (2-4)
- (*) Linguistic trees in a relational database (particularly for LASSY project). (2-4)
- (*) Hierarchies on a database. (2-4)
- A 'research output' database for a research institute.(2–4)
- A quiz database (1–3).
- A meeting/event scheduler. (2–4)
- An electronic voting system (1–2).
- A Personal library management system (1–2).
- A Personal financial-accounting software (1–2).

. . . or your own proposal.

(*) — projects with customers: your work may be used for solving a real-world problem.

Projects and deadlines



15 Team formation and project choice.

28 Initial project report.

16 (optional) pre-final report. If you do, you will get comments and suggestions by 2012-01-23.

31 Deadline for the finalized project/report. **No extensions!**

Project evaluation

- Good database design
- Secure and efficient web programming
- You will need to submit two reports:
  - Initial design report (3 to 8 pages), which should include,
    - a summary of the project requirements
    - your Initial database design (e.g., E-R design, DB schema)
    - type(s) of users, typical queries expected, security (authentication, authorization) requirements.
  - You will get feedback on your design based on this report.
  - Final report (5 to 10 pages), which can include the some of the above, but also,
    - a brief 'user guide'
    - final state of the project and possible future work
  - Clarity of the reports will contribute to your final score.
  - All project documentation has to be in English, but you will not loose points for language mistakes.

PHP: introduction

- PHP is especially targeted for web programming.
- Works with most web-server software.
- PHP is very popular, you can easily find examples and documentation on the Internet.
- Most web hosting services offer PHP.
- PHP syntax is similar to C/C++/Java.

PHP: a first example

```
<html>
<head></head>
<body>
<?php
    /* This is a first example
     * in PHP. This code is
     * actually pointless.
     */

    echo "Hello World!<br>\n";

    // We can also have short
    // comments.
?>
</body>
```

- PHP code (typically) goes inside an HTML docuemnt.
- All PHP code goes in between `<?php` and `?>`
- All statements end with a semicolon ';'.
- C-style comments are allowed.

Variables in PHP

- PHP variables start with a `$`.
- Variables are automatically typed (you do not need to declare them in advance).
- Variable assignement can be done via `=` operator: `$i = 1;`.
- The following basic types exist:
  - Boolean: `true` (1) or `false` (0).
  - Integer: `-12`, `12`, `014` (octal), `0xC` (hexadecimal).
  - Floating point: `1.2`, `1.2e10`.
  - String: `'abc'`, `"abc"`.
- PHP will try to be smart if you mix and match types, but you can also do C-style type casting.
- You can use debugging function `var_dump()` if you are unsure about the type of an expression (and if it matters).

PHP: basic operators

- Arithmetic operators: `+`, `-`, `/`, `*`, `%`.
- Increment/decrement: `++`, `--`.
- Comparison operators: `==` (equal), `!=` or `<>` (not equal), `<` (less than), `>` (greater than), `<=` (less than or equal), `>=` (greater than or equal).
- More comparison operators: `===` (identical), `!==` (not identical).
- Logical operators: `&&` (and), `||` (or). `!` (not).
- String concatenation: `.`
- Assignment: `=`, `+=`, `-=`, `/=`, `*=`, `%=`, `.=`.
- Operator precedence is similar to C-family languages. When unsure: use parentheses.

PHP operators: some examples

```
<?php
$a = 1;                 // $a = 1
$b = $a + 5;            // $b = 6
$a += 5;                // $a = 6
$a /= 3;                // $a = 2
$b %= 4;                // $b = 2
$a = 5 + 2 * 4;         // $a = 13
$a = (5 + 2) * 4;       // $a = 28
++$a                    // $a = 29
$b = $a + '3 apples'    // $b = 32
$b = $a + 10.2          // $b = 42.2
$b = ($a == 29)         // $b = true
$b = ($a == '29')       // $b = true
$b = ($a === '29')      // $b = false
$b = '3 apple' . 's'    // $b = '3 apples'
$b .= ' and 2 pears'    // $b = '3 apples and 2 pears'
?>
```


PHP strings

- String constants can be enclosed in single (') or double (") quotes.
- Strings in the single quotes are interpreted literally.
  `$a = 'ABC'; $b = '$a'; // $b is '$a'`
- Strings in double quotes are expanded.
  `$a = 'ABC'; $b = "$a"; // $b is 'ABC'`
- You can also use so called 'here documents'.
- Individual characters in a string can be accessed (and replaced) using array notation. Index starts at `0`.
  `$a = 'ABC'; $b = $a[1]; // $b is 'B'`
- For larger substrings you can use `substr()` or `substr_replace()`.

PHP arrays

All PHP arrays are associative arrays.

- Arrays can be created using `array()`:
  `$prime = array(2,3,5,7,11);`
  or if `$prime` does not exist yet,
  `$prime[0] = 2;prime[]=3;` ...
  will also create an arrray.
- Using simple arrays:
  `$a = prime[0] + prime[1];`
- Array keys can be any type, and can be specified with the notation:
  `$prime = array('first' => 2, 'second' => 3, 'third' => 5);`
  `$prime['first'] = 2; prime['second'] = 3` ...

PHP classes and objects

PHP allows object-oriented programming syntax is similar to C++/Java:

```
class user {
    public username;
    private password;
    public function check_password($pwd){
        return ($this->password === $pwd);
    }
    public function set_password($pwd){
        $this->password = $pwd;
    }
}
$u = new user;
```

Other OO constructs, such as inheritance, are also supported.

PHP control structures

- if-else

```
if (condition) {
    // do this if condition is true
} else {
    // do that if condition is false
}
```

- switch-case

```
switch ($var) {
    case 1: echo "\$var is 1";
    break;
    case 1.2: echo "\$var is 1.2";
    case 'a': echo "\$var is 1.2 or 'a'";
    break;
    default:
       echo "\$var is not 1, 1.2, or 'a'";
}
```


PHP loops

- while loop

```
while (condition) {
    // do this while condition is true
}
```

- do–while

```
do {
    // do this once, and while cond. is true
} while (condition);
```

- for

```
for ($i = 0; $i < $N; $i++) {
    // do this $N times, for $i = 0 ... $N-1
}
```


PHP looping over an array

- If we are not interested in keys

```
foreach ($arr as $val) {
    // do this for each $val in $arr
}
```

- If we also need the keys

```
foreach ($arr as $key => $val) {
    // do this for each ($key => $val) pair
}
```


PHP functions

We have already seen examples while discussing classes. Here is another:

```
function length_array($arr)
{
    $i = 0;
    foreach ($arr as $val) {
        $i++;
    }
    return $i;
}
```

Arguments are passed by value, if you want to pass them by reference, you need to prepend with `&` in definition. For example: `function array_length(&$arr)`.

Scope of variables

- Except in functions (and classes), all variables are global, and share the same scope.
- In functions, only the variables declared inside the function is accessible, and they are local.
- To access a global variable inside a function, it has to be declared as `global`.

```
<?php
$a = 0;

if (isset($a)) {
    $a = 1;
    $b = 0;
}
?>

<?php
echo $a; // outputs 1

function test_function()
{
    global $b;
    $a = $b;
    $b = 1;
    echo "$a"; // outputs 0
}

echo $a; // outputs 1
echo $b; // outputs 1
?>
```


File I/O

- C/stdio like `fopen()` – `fread()`/`fwrite()` – `fclose()` interface can be used to interact with files.
- You can also give a URL to `fopen()`.
- A large number of (convenience) I/O functions exist. For example, `file_get_contents()` would do instead of the code on the right.

```
define('BUFSIZE', 4046);
$data = '';
$fh = fopen("/tmp/test.file", "r");
if ($fh) {
    echo "Cannot open file.\n";
} else {
    while (!feof($fh)) {
        $data .= fread($fh, BUFSIZE);
    }
    fclose($fh);
}
```


PHP: summary so far

- PHP is a general purpose language, particularly designed for web programming.
- We just mentioned the basics without interesting bits.
- We will return to PHP for
  - Interactive web programming
  - Database access

Version Control Systems

A version control system (or, *revision control* or *source control* system) is commonly used in software development.

A VCS,

- Records a history of all changes to all files under VC.
- Allows going back in time, to any version of the software.
- Allows inspecting which change happened when.
- Allows maintaining multiple branches of the same software without multiple copies.
- Allows sandboxing: you can try (experimental) changes without disrupting the 'working copy'.
- Facilitates team work.
- Can also be used for other purposes, for example, web pages, documents . . .

VCS: some terms

repository is the database where the VCS stores the code, versions and associated information.

working copy is where the programmer makes the changes.

tag is a name given to the state of the repository at a certain time.

branch is a *virtual* copy of the whole repository for a specific purpose.

check-in operation updates the repository using the working copy.

merge brings code in different branches (possibly from different programmers) together.

conflict may occur during a merge, if same segment of the code was changed in different ways.

Some VC systems

RCS Dates back to early 1980's. Typically single user, single system. Keeps track of single files.

CVS Client-server system with support for multiple users on a network. Server side manages the central repository, client side keeps the users' working copies.

SVN An improvement over CVS.

GNU arch, mercurial, bazaar, git . . . are distributed VCSs. Every working copy keeps a repository. The repository for each programmer is local, but can be merged using different methods.

Git: introduction

- Git is a distributed VCS.
- Developed (primarily) for Linux kernel, but used by many projects.
- Available for almost all operating systems.
- Can be shared through HTTP, SSH, git, rsync, email . . .
- Many (somewhat) free software-hosting providers for projects using git. Just a few: GitHub, Gitorious, Bitbucket . . .

Git: starting a new repository

```
git init
```

- Initializes an empty git repository in the current directory (you need to create one if it does not exist yet).
- The all repository lives in `.git` subdirectory.
- The only file of interest (probably) here is `.git/config`.

```
mkdir testproject
cd testproject
git init
$EDIT index.php #=====>
```

```
<html>
<head></head>
<body>
<?php
    echo "Hello Wrold!\n";
?>
</body>
</html>
```

Git: recording changes in the repository

`git add` and `git commit`

- Git has a two-stage commit process. First you need to `add` the changes you want to commit. (This adds the change to a 'cache', called 'index', but we are not concerned with that.)
- Then, `commit` updates the repository with the changes.

```
git add index.php
git commit -m "added index.php"
```

- `-m` option to `commit` gives a short comment. Otherwise, git fires up a text editor to write a comment.
- Comments are important. Do not skip and be descriptive.

Git: inspecting changes

`git diff`

```
# found a bug!
# edit index.php, change 'Wrold' to 'World'
git diff
--- a/index.php
+++ b/index.php
@@ -4,7 +4,7 @@
 <?php
-    echo "Hello Wrold!";
+    echo "Hello World!";
 ?>
```

- Without options, `diff` will give you differences between the working copy and the `HEAD` of the repository for all files.
- You can inspect differences for any set of files between any two states of the repository using `diff` (more on this later).

Git: checking the status

`git status`

```
git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   index.php
#
no changes added to commit (use "git add" and/or "git commit -a")
```

- `status` gives you a summary of the current state of your working copy.
- Let's commit our bug-fix:

```
git commit -a -m "fixed a typo"
```


Git: logs

`git log`

```
git log
commit 933f1de6b727d7816df9420aa4f0c85c00455d19
Author: Cagri Coltekin <c.coltekin@rug.nl>
Date:   Sun Nov 6 15:59:31 2011 +0100

    fixed a typo

commit 7742fd6ff2a90372bf9545732a9e6923588d3052
Author: Cagri Coltekin <c.coltekin@rug.nl>
Date:   Sun Nov 6 15:31:56 2011 +0100

    added inex.php
```


Git: working with branches

`git branch` and `git checkout`

- Branches are fast and cheap: use them!
- Branches can be used for maintaining long-term different versions of software, or short term experimental changes.

```
$ git branch experimental       # we are about to do a major change create
$ git checkout experimental     # a new branch and switch to it
Switched to branch "experimental"
$ git status # let's see if everything is as expected
# On branch experimental
nothing to commit (working directory clean)
$ $EDITOR index.php # change 'Hello' to 'Hi'
$ $EDITOR LICENSE   # also add a LICENSE file
$ git status        # check the status again
# On branch experimental
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#       modified:   index.php
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#       LICENSE
no changes added to commit (use "git add" and/or "git commit -a")
```


Git: working with branches (2)

```
$ git add LICENSE   # add the new file
$ git add index.php # add the changes to index.php
$ git status        # checking once more doesn't hurt
# On branch experimental
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   LICENSE
#       modified:   index.php
#
$ git commit -m "Major change in index.php + added LICENSE"
$ ls
index.php  LICENSE
$ grep World index.php
    echo "Hi World!";
$ git checkout master
$ ls
index.php
$ grep World index.php
    echo "Hello World!";
```

- Note: your commits should better be atomic.

Git: merging branches

`git merge`

```
$ git merge experimental
Updating 933f1de..e07aa49
Fast forward
 index.php |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
 create mode 100644 LICENSE
```

- this merge succeeds with no conflicts
- now both branches are identical. We can delete the `experimental` branch if we like with the command `git branch -d experimental`.
- if there were conflicts, we'd need to resolve them manually (helpful tools exist).

Git: interacting with other repositories

`git clone`, `git pull` and `git push`

- `clone` makes a complete copy of a repository.
- As well as local files, a repository can be accessed (and cloned) through a number of ways. Including, `ssh`, `http`.
- This allows git to be used in a client–server-like setup.
- Once you cloned an original source, you can use `git pull` to receive updates, and `git push` to push your changes.
- You can `pull` from or `push` to multiple repositories.
- A number of software hosting sites are primarily dedicated for git.

Git: graphical user interfaces

`git gui`, `gitk`, `qgit` ...

- the native gui `git gui` allows you to do most of these operations by point&click.
- a few others exist: `gitk`, `qgit`, `Git Extensions` (for Windows), `GitX` (for Mac) ... See `http://git-scm.com/tools` for more.
- Hosting sites, e.g., GitHub, also provide some web-based functionality.

Git (or other VCS) in your projects

- This is just a summary, git (and most other VCSes) provide many possibilities that makes your life easier in the long run.
- More documentation and pointers to good tutorials can be found at `http://git-scm.com/documentation`.
- Take it seriously in your projects.
- The time you invest to learn to use a VCS will pay off.
- You are required to point me to a repository that I can inspect for your projects.

Summary & next week

Today:

- Some review of PHP.
- A quick introduction to software version control using git.

Next week:

- Tuesday: project/team selection. Demo, Q&A on PHP/git.
- Wednesday: back to databases.