Databáze pro začátečníky
1. díl - Úvod do databází v PHP pro úplné začátečníky

Vítejte v prvním dílu seriálu tutoriálů o práci s MySQL databází v PHP **pro úplné začátečníky a bez objektově orientovaného programování**. MySQL je nejpoužívanější databáze, která je zadarmo a máte ji k dispozici snad na každém webhostingu. V několika lekcích zde vytvoříme funkční příklady a nakonec také jednoduchý blog s editorem článků.

Úvod

Seriál se budu snažit psát co nejjednodušeji. Budu předpokládat alespoň minimální základy syntaxe PHP. U MySQL databáze si v seriálu popíšeme nutné minimum, čili zde vůbec žádné znalosti nepotřebujete. Znalosti OOP také nepotřebujete.

Pro zájemce jsou zde samozřejmě i podrobnější seriály o MySQL databázi a podrobnější seriály o objektově orientovaném programování. Tento seriál je vlastně taková velmi odlehčená verze, aby si databáze mohl v PHP zkusit úplně každý.

Proč použít databázi

Jako uživatelé nějakého webu chceme určitě měnit jeho obsah (napsat článek, komentář, přidat zboží do eshopu, zaměstnance do firmy a podobně). Bylo by opravdu nepohodlné bušit pro každý článek novou HTML stránku a tu nahrávat na server. V článcích (produktech, zaměstnancích...) by se navíc nedalo vyhledávat a při větším počtu by to bylo velmi nepřehledné. Mnohem lepší způsob je naprogramovat si v PHP editor článků, který články ukládá do databáze a čtenářům je poté z databáze zobrazuje.

Možná vás napadlo, že k tomu vlastně ani nepotřebujeme nějakou databázi. Data bychom stejně dobře mohli ukládat editorem do nějakých textových souborů nebo něčeho podobného. Určitě by to nějak fungovalo, nebo ne?

Databáze není jen úložiště dat. Jedná se o velmi sofistikovaný a odladěný nástroj, který za nás řeší spoustu problémů a zároveň je extrémně jednoduchý k použití. S databází totiž komunikujeme jazykem SQL, kterým jsou v podstatě lidsky srozumitelné věty. S databází si opravdu píšeme stylem "vyber mi uživatele, jehož email je franta@seznam.cz" nebo "vlož do uživatelů nového se jménem Jan a emailem jan@seznam.cz". Případně můžeme napsat třeba i toto: "vyber mi jméno uživatele s emailem franta@seznam.cz a jeho články, seřazené podle data vydání sestupně". Budete překvapeni, jak jednoduché to je.

Databáze za nás řeší ještě spoustu dalších problémů ohledně ukládání dat, které bychom sami asi těžko překlenuli. Zde je však nebudeme rozvádět.

Databáze je tedy taková černá skříňka, která za nás nějakým způsobem řeší vše okolo dat a my se o to nemusíme starat a pouze ji jednoduše používáme. V dnešní době se již jinak s daty prakticky nepracuje.

Relační databáze

MySQL je tzv. relační databáze. Tento pojem označuje databázi založenou na tabulkách. Každá tabulka obsahuje položky jednoho typu. Můžeme mít tedy tabulku *uzivatele*, další tabulku *clanky* a další třeba *komentare*. Tabulky se většinou pojmenovávají spíše v jednotném čísle, pro účely tohoto kurzu mi ale přišlo logičtější použít plurál.

Databázovou tabulku si můžeme představit třeba jako tabulku v Excelu. Tabulka *uzivatele* by mohla vypadat asi takto:

Jméno	Příjmení	Datum narození	Počet článků
Jan	Novák	11.3.1984	17
Tomáš	Marný	1.2.1989	6
Josef	Nový	20.12.1972	9
Michaela	Slavíková	14.8.1990	1

Položky (konkrétně zde uživatelé) ukládáme na jednotlivé řádky, sloupce pak označují atributy (vlastnosti, chcete-li), které položky mají. MySQL databáze je typovaná, to znamená, že každý sloupec má pevně stanovený datový typ (číslo, znak, krátký text, dlouhý text...) a může obsahovat hodnoty jen tohoto typu. Pokud chceme s relační databází rozumně pracovat, každý řádek v tabulce by měl být opatřený unikátním identifikátorem. U uživatelů by to mohlo být třeba rodné číslo, mnohem častěji se však používají identifikátory umělé a to tak, že uživatele prostě očíslujeme. K tomu se dostaneme později.

Slovo *relační* označuje vztah (anglicky relation), které mohou být mezi entitami v databázi, ale tím se nebudeme zatím zabývat.

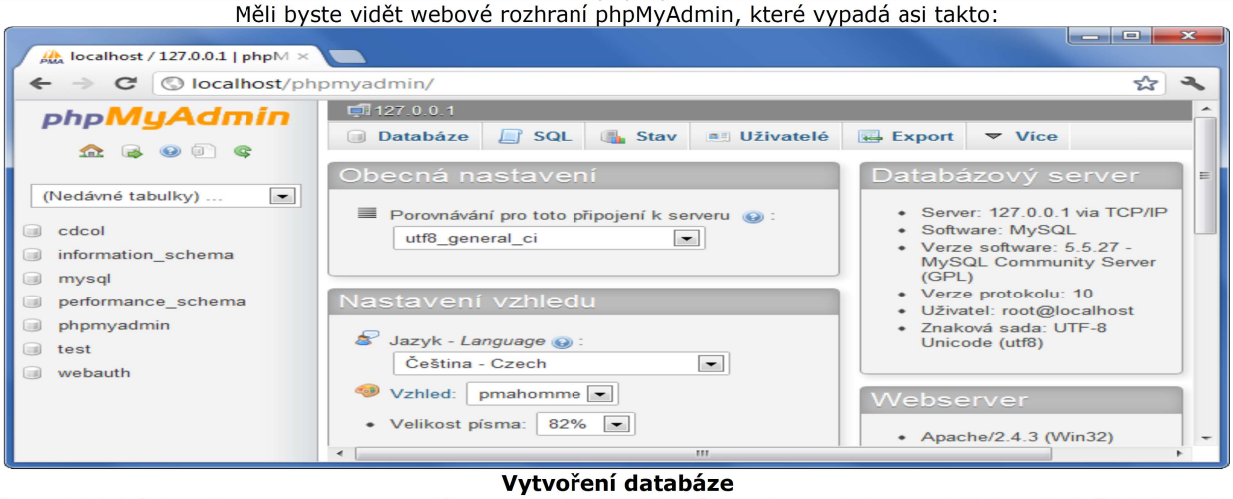
Potřebné nástroje

Začněme tedy. Potřebovat budeme nakonfigurovaný webserver Apache a databázi MySQL. Předpokládám, že jste již někdy spustili nějaký PHP skript a že prostředí tedy máte k dispozici, případně zde je návod na instalaci PHP a MySQL.

phpMyAdmin

phpMyAdmin je nejpoužívanější prostředí pro práci s MySQL databází. MySQL s phpMyAdminem naleznete na každém webhostingu a pokud se někdy dostanete k cizímu projektu, je velmi pravděpodobné, že bude postaven právě na těchto technologiích. K phpMyAdmin se dostanete přes administrační rozhraní vašeho webhostingu. Na localhostu má obvykle adresu: localhost/phpmyadmin/

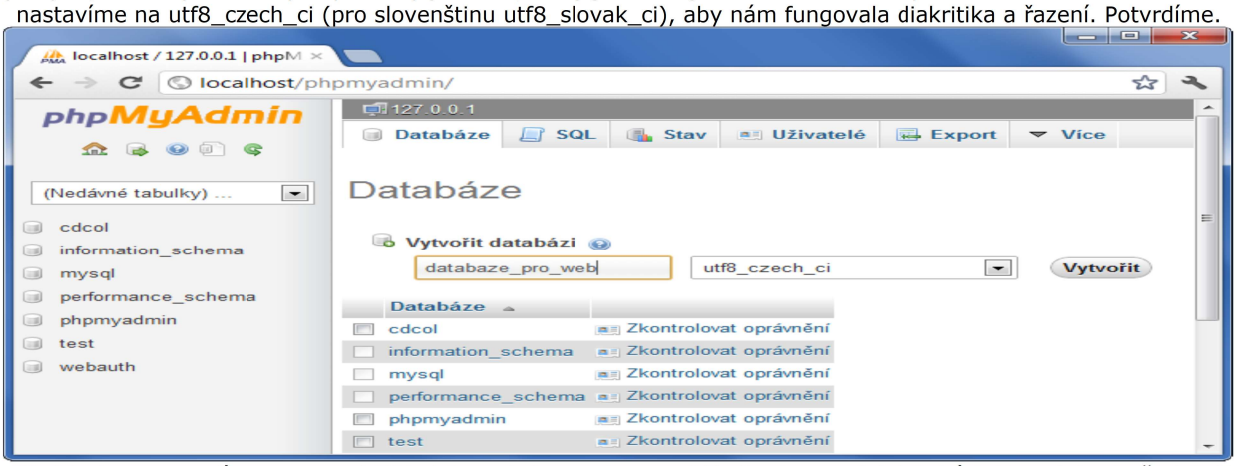
Měli byste vidět webové rozhraní phpMyAdmin, které vypadá asi takto:



Vytvoření databáze

Vytvořme si databázi, se kterou budeme v seriálu pracovat. Obvykle nám bude pro jeden projekt (web) stačit jedna databáze. Tento krok za nás již na webhostingu téměř vždy udělali a dostaneme k dispozici často právě jednu databázi, která se obvykle jmenuje stejně, jako náš účet (tedy třeba něco jako mojewebovastrankacz1). Na localhostu si ji musíme vytvořit sami.

Klikněme v phpMyAdmin nahoře na záložku Databáze. Vyplníme název databáze (např. databaze_pro_web). V databázích je zvykem pojmenovávat položky bez diakritiky, malými písmeny a s podtržítkovou notací. Snad vám je jasné, proč není diakritika dobrý nápad, za velkými a malými písmeny je Linux, který je rozlišuje a většina serverů právě na Linuxu běží. Porovnávání nastavíme na utf8_czech_ci (pro slovenštinu utf8_slovak_ci), aby nám fungovala diakritika a řazení. Potvrdíme.



PhpMyAdmin na pozadí vygeneroval SQL dotaz z toho, co jsme naklikali a ten poslal databázi. Vypadal asi nějak takto:

CREATE DATABASE databaze_pro_web CHARACTER **SET** utf8 **COLLATE** utf8_czech_ci;

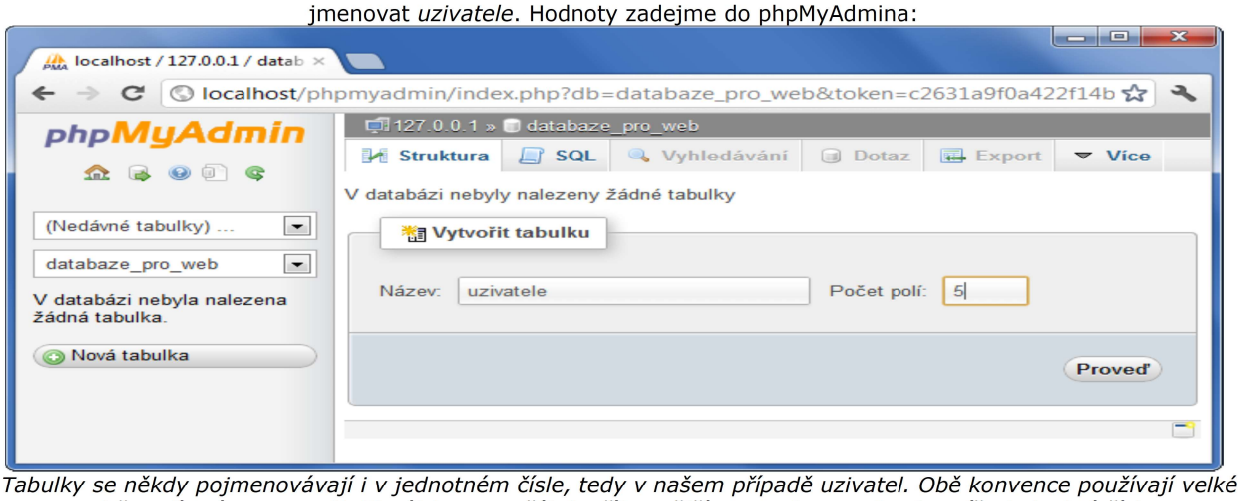
Ale pamatovat si ho nemusíte, databáze a tabulky budeme vždy klikat přes phpMyAdmin. Databázi máme připravenou, tvorbu tabulky si necháme na příště.

2. díl - První databázová tabulka a MySQL ovladače v PHP

V minulém dílu seriálu tutoriálů o databázích v PHP pro úplné začátečníky jsme si udělali krátký úvod do databází a vytvořili si databázi pro náš web. V dnešním dílu si vytvoříme tabulku uzivatele a vložíme do ni nějaká data.

Tabulka

Databázi si v levém sloupci otevřete. phpMyAdmin nám nabízí vytvoření tabulky. Vzpomeneme si na příklad tabulky uživatelů, co jsme si ukázali v minulém dílu. Měla sloupce jméno, příjmení, datum narození a počet článků. Již jsme nakousli, že by každá tabulka měla mít sloupec, jehož hodnota je pro každý řádek unikátní. Sloupců bude tedy dohromady 5, tabulka se bude jmenovat *uzivatele*. Hodnoty zadejme do phpMyAdmina:



Pozn.: Tabulky se někdy pojmenovávají i v jednotném čísle, tedy v našem případě uzivatel. Obě konvence používají velké firmy a obě mají svá pro a proti. Plurál je pro začátek příjemnější, v praxi se preferuje spíše jednotné číslo.

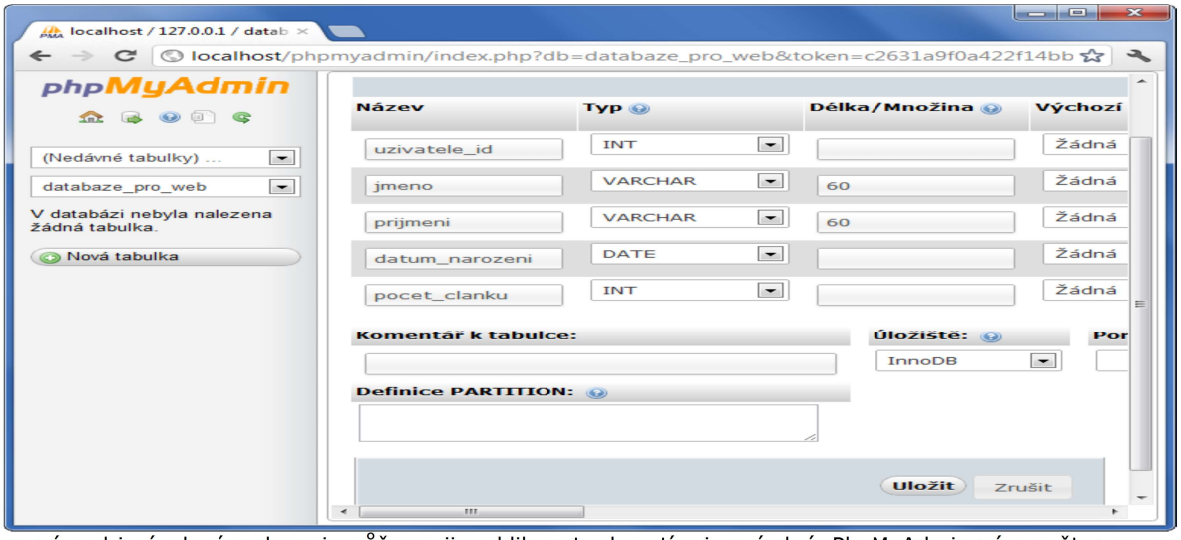
Otevře se nám opravdu hodně polí, ale těch se vůbec nelekejte. První sloupec jsou jména sloupců tabulky. Vyplníme pod sebe tedy názvy našich sloupců, což jsou: uzivatele_id, jmeno, prijmeni, datum_narozeni, a pocet_clanku. Id se mi osvědčilo pojmenovávat s prefixem tabulky, aby se předešlo následným kolizím názvů, ale není to nutné.

Přesuňme se k 2. sloupci, kde jsou datové typy jednotlivých sloupců tabulku. Přednastavený máme INT, což jsou celá čísla. Typů je opravdu mnoho, ale my si dlouho (asi až do konce seriálu) vystačíme jen s několika. uzivatele_id ponecháme na INT, jmeno a prijmeni nastavíme na VARCHAR, to je krátký text. Datum narození na DATE. Poslední, pocet_clanku, bude typu INT. Časem si popíšeme i další datové typy, ale teď vám s nimi nebudu motat hlavu

3. sloupec Délka/množina má smysl jen u VARCHARU a udává maximální počet jeho znaků, jméno i příjmení nastavíme na 60 znaků.

Dále máme další sloupce, které pro nás však nejsou tak důležité a nebudeme je vyplňovat. Poslední, co vyplníme, je sloupec Klíč u sloupce s uzivatele_id. Klíč zde nastavíme na PRIMARY a zaškrtneme pole vedle, které je nadepsáno A_I (jako Auto Increment). Tímto jsme sloupec uzivatele_id nastavili jako tzv. primární klíč tabulky. Klíče (někdy indexy) nám umožňují identifikovat položku v tabulce. Takový primární klíč by měla mít každá tabulka (i když teoreticky nemusí). Když budeme chtít uživatele např. vymazat, vymažeme ho podle tohoto klíče (tedy podle uzivatele_id). Kdybychom ho mazali podle jména, smazali bychom několik položek, protože třeba Janů Nováků tam může být více. Podle uzivatele_id vymažeme vždy jen toho jednoho. Zaškrtnutí Auto Increment způsobilo, že se bude hodnota uzivatele_id automaticky navyšovat a uživatelé se budou postupně číslovat.

Mimochodem, všimněte si možnosti Přidat pole, to je pokud jich na začátku zadáme málo a při návrhu tabulky zjistíme, že potřebujeme další. Tabulku uložíme.



Tabulka se nám objeví v levém sloupci, můžeme ji rozkliknout, ale zatím je prázdná. PhpMyAdmin nám opět vygeneroval SQL dotaz, který vypadá asi takto:

```
CREATE TABLE uzivatele (
uzivatele_id int AUTO_INCREMENT,
jmeno varchar(60),
prijmeni varchar(60),
datum_narozeni date,
pocet_clanku int,
PRIMARY KEY (uzivatele_id)
);
```

Opět si ho nemusíme pamatovat.

MySQL ovladač v PHP

K databázi se nyní připojíme z PHP a vložíme do ní nějaké uživatele. Abychom tak mohli učinit, potřebujeme databázový ovladač. To je v PHP překvapivě trochu problém a právě na tomto bodě ztroskotá většina začátečníků a od databází zbytečně odejde. Databázové ovladače jsou v PHP hned 3:

mysql

S databází ovladač mysql komunikoval pomocí funkce mysql_query(). Jelikož tento ovladač používal staré přístupy, které jsou poměrně nebezpečné (např. bylo nutné ručně ošetřovat parametry dotazů), byl od PHP 5.5 označen jako zastaralý a jeho použití vyvolá chybovou hlášku. Další verze PHP ho již nebudou obsahovat. Tento ovladač se bohužel stále učí v některých knihách a na některých školách, na což si dejte pozor. Používat byste ho rozhodně neměli.

mysqli

mysqli označuje MySQL Improved. Jedná se o hybridní ovladač, který šlo používat jak objektově, tak procedurálně (funkcemi). Procedurální funkce však z PHP manuálu nějak vymizely a používá se výhradně objektově. Parametry se do dotazů předávají velmi nepohodlně.

PDO

PDO je zkratka PHP Database Objects. Jedná se o nejnovější a velmi kvalitní objektový ovladač, který se jednoduše používá a podporuje kromě MySQL ještě několik databází. Funkcím, které začínají mysql, je již v PHP lepší se úplně vyhnout.

Wrapper

Jak je vidět, tak v PHP s databází neobjektově pracovat v podstatě nelze. Co však lze je použít tzv. wrapper. Wrapper by se dal přeložit jako obal. Někdo pro vás obalí objektové funkce ovladače PDO tak, abyste jim rozuměli a mohli je používat bez toho, aniž byste vůbec tušili co to objekt je. Právě tento přístup zvolíme v našem seriálu.

První příklad - vložení uživatele do databáze

Vytvořte si nějaký PHP projekt a pojmenujte ho například TestDatabaze. Stáhněte si přílohu dnešního článku a vytáhněte si z ní soubor s wrapperem (Db.php) a vložte ho do složky s projektem (aby byl ve stejné složce jako index.php). Svůj index.php nyní upravte do následující podoby:

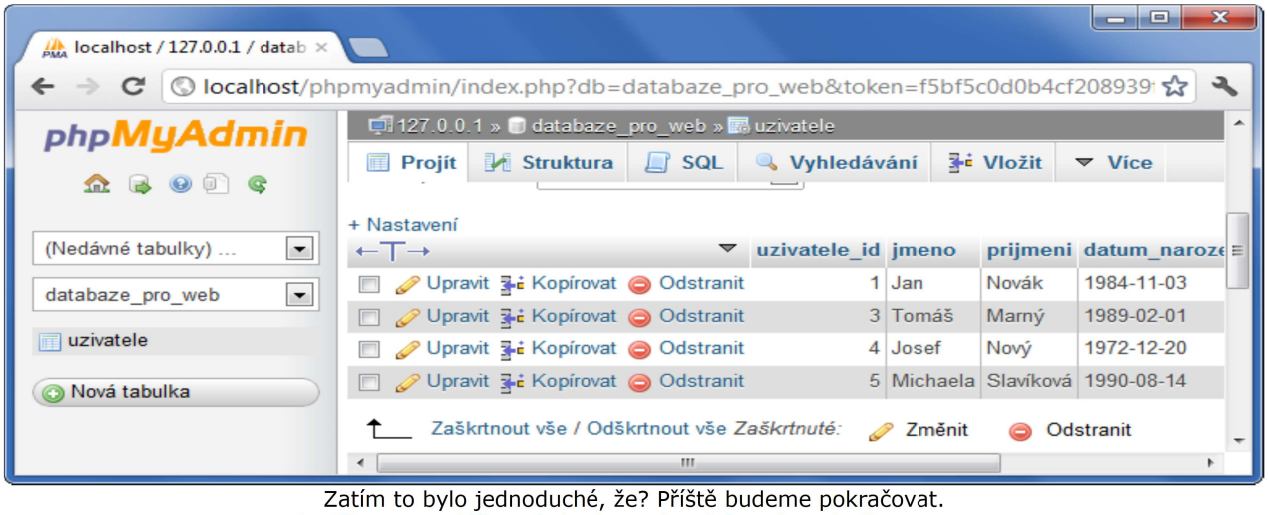
```
// Načtení wrapperu
require_once('Db.php');
Db::connect('127.0.0.1', 'databaze_pro_web', 'root', '');
Db::query('
INSERT INTO uzivatele (jmeno, prijmeni, datum_narozeni, pocet_clanku)
VALUES ("Jan",  "Novák",  "1984-11-03", 17)
');
echo('OK');
```

Příklad načte databázový wrapper ze souboru Db.php. Všechny funkce wrapperu nám jsou nyní přístupné pod třídou Db a voláme je jako Db::nazevFunkce().

Jako první se k databázi připojíme pomocí funkce Db::connect(). Zde zadáme název hostitele, název databáze, uživatelské jméno a heslo. Na localhostu jsou většinou údaje jako v příkladu výše, na produkci vám tyto údaje sdělí webhosting.

Další funkcí wrapperu je Db::query(), která na databázi spustí dotaz v jazyce SQL. SQL dotaz se zadává jako textový řetězec a pro přehlednost jsem ho napsal na 2 řádky, obvykle se to tak dělá. Když dotaz přeložíme do češtiny, říkáme databázi: "Vlož do uživatelů do sloupců jmeno, prijmeni, datum_narozeni_pocet_clanku hodnoty Jan Novák 1987-11-13 17". Anglicky (v jazyce SQL) to vypadá jak je uvedeno výše. Všimněte si, že textové řetězce píšeme do uvozovek (stejně jako v PHP) a datum zadáváme v americkém tvaru rok-mesic-den. Čísla píšeme jak jsou. **Pozor! Nikdy nevkládejte PHP proměnné do textu dotazu! Vystavili byste se tak velkému bezpečnostnímu riziku, které si vysvětlíme příště**.

Skript si spusťte a to klidně několikrát s různými daty. Nyní přejdeme do PHPMyAdmin, rozklikneme tabulku uzivatele a zobrazíme si je (tlačítko Projít nahoře). Vidíme, že jsou v databázi opravdu vložení:



Zatím to bylo jednoduché, že? Příště budeme pokračovat.

3. díl - Formulář a výpis dat z databáze do tabulky v PHP

V minulém dílu seriálu tutoriálů o databázích v PHP pro úplné začátečníky jsme se připojili k databázi a vložili do ní několik uživatelů. V reálných aplikacích se uživatelé vkládají pomocí formuláře. Přesně to se dnes naučíme a také se naučíme vypsat uživatele z databáze do HTML tabulky. Budeme pokračovat ve stylu co nejjednoduššího kódu.

Formulář

Pro vkládání uživatelů do databáze si vytvoříme jednoduchou HTML stránku s jedním formulářem. Náš vkládací dotaz upravíme tak, aby vkládal hodnoty z formuláře. Uveďme si kompletní kód registrační aplikace:

```
<!DOCTYPE html>
<html lang="cs-cz">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>Registrace uživatele</title>
</head>

<body>

<h1>Registrace uživatele</h1>

<?php

require_once('Db.php');
Db::connect('127.0.0.1', 'databaze_pro_web', 'root', '');
if ($_POST)
{
$datum = date("Y-m-d H:i:s", strtotime($_POST['datum_narozeni']));
Db::query('
INSERT INTO uzivatele (jmeno, prijmeni, datum_narozeni)
VALUES (?, ?, ?)
', $_POST['jmeno'], $_POST['prijmeni'], $datum);

echo('<p>Byl jste úspěšně zaregistrován.</p>');
}
?>

<form method="post">
Jméno:<br />
<input type="text" name="jmeno" /><br />
Příjmení:<br />
<input type="text" name="prijmeni" /><br />
Datum narození:<br />
<input type="text" name="datum_narozeni" /><br />
<input type="submit" value="Registrovat" />
</form>

</body>
</html>
```

Kód je stále velmi krátký. Vysvětleme si ho. Co se týče HTML, tak tam by mělo být vše jasné. Formulář je jednoduchý a obsahuje 3 pole (jméno, příjmení, datum narození) a odesílací tlačítko. Protože form postrádá atribut action, odešle se na tu samou stránku.

Ve stránce je rovněž PHP skript, který se připojí k databázi. Podmínkou otestuje, zda se odeslala nějaká data formulářem. Pokud ano, vykoná SQL dotaz, který data do databáze vloží. Všimněme si třech věcí:

1. Datum musíme převést z českého formátu (tak, jak ho zadal do políčka uživatel, např. 15.1.1989) do formátu MySQL (např. 1989-15-1). To za nás udělá dvojice funkcí str_to_time() a date().
2. SQL dotaz je velmi podobný tomu z minulého dílu. Již nevkládáme do všech sloupců, ale jen do třech. Do sloupce pocet_clanku se vloží výchozí hodnota, tedy 0.

3. Do dotazu zde již potřebujeme vložit proměnné z PHP (konkrétně od uživatele z $_POST). **A nyní pozor: Proměnné NIKDY! nevkládáme přímo do dotazu! Kdyby uživatel zadal místo jména nějaký SQL příkaz, tak by se totiž do dotazu vložil a provedl na naší databázi! Místo parametrů v dotazu vždy píšeme otazníky a potom parametry předáme ve stejném pořadí jako další parametry funkce Db::query()! Tuto chybu zde neustále opakují začátečníci stále a stále dokola, hazardujete se svými daty a daty vašich uživatelů!**

Ukažme si ještě raději, jak se to **nemá** dělat:

```
// TENTO KÓD JE VELMI NEBEZPEČNÝ!
Db::query('
    INSERT INTO uzivatele (jmeno, prijmeni, datum_narozeni)
    VALUES ("'. $_POST['jmeno'] . '", "' . $_POST['prijmeni'] . '", "' . $datum . '")');
```

Proměnné jsou vložené přímo v SQL dotazu. Když uživatel napíše do políčka pro jméno tento řetězec:

```
", ""); DELETE FROM uzivatele; --
```

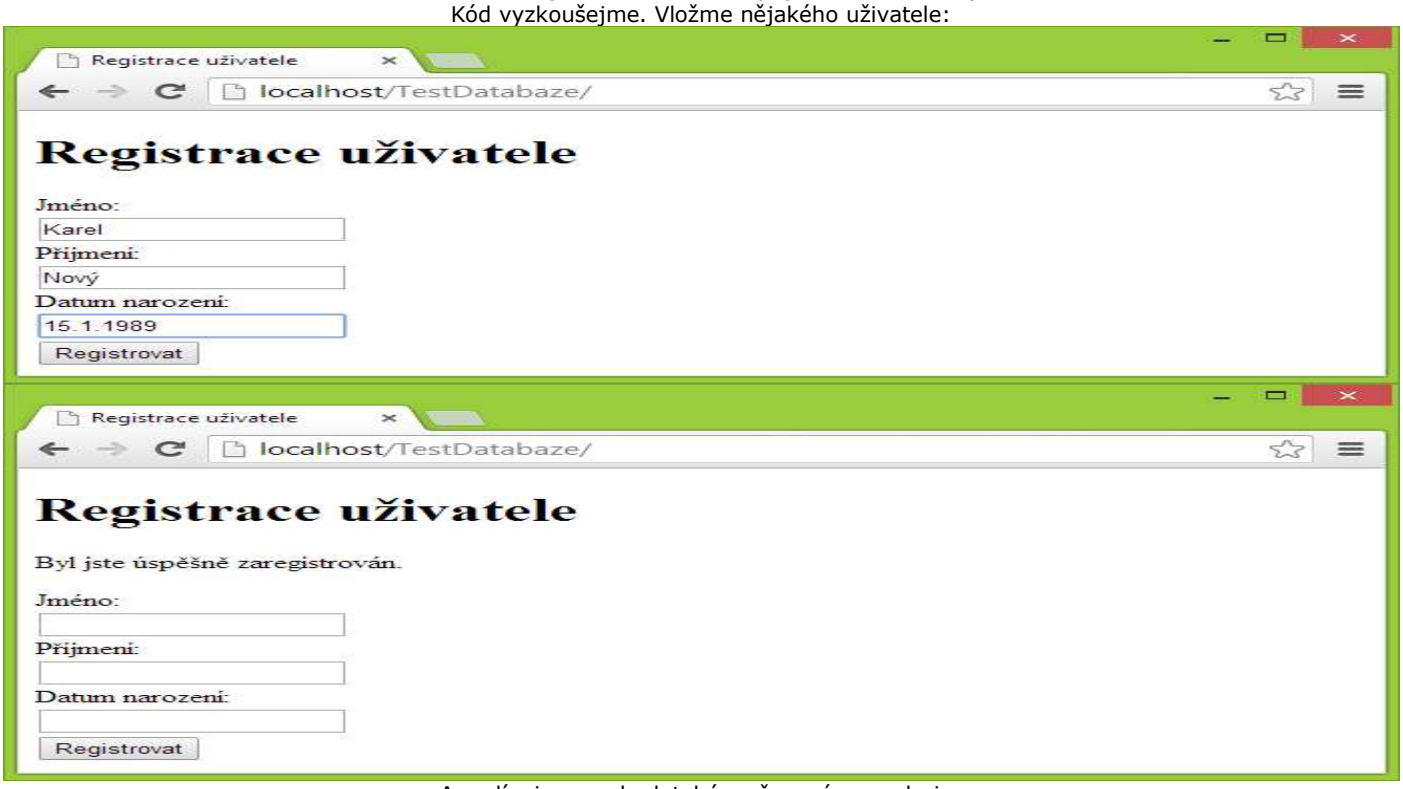
Smaže nám všechny uživatele v databázi, protože co napsal se vloží přímo do dotazu a příkaz se vykoná. Tomuto útoku se říká SQL injection. Zrovna proti tomuto případu je náš wrapper imunní, jelikož jsou v něm určitá nastavení, která zrovna tento typ injekce nepovolí. Nejedná se však o výchozí nastavení a v žádném případě nezastaví další typy injekcí.

Kdykoli chceme do dotazu vložit nějaký parametr, použijeme otazník a napíšeme ho mimo dotaz! Databáze si tam potom parametr sama a bezpečně dosadí, nikdy to nedělejte za ni. **Nikdy nepřerušujte řetězec s SQL dotazem.**

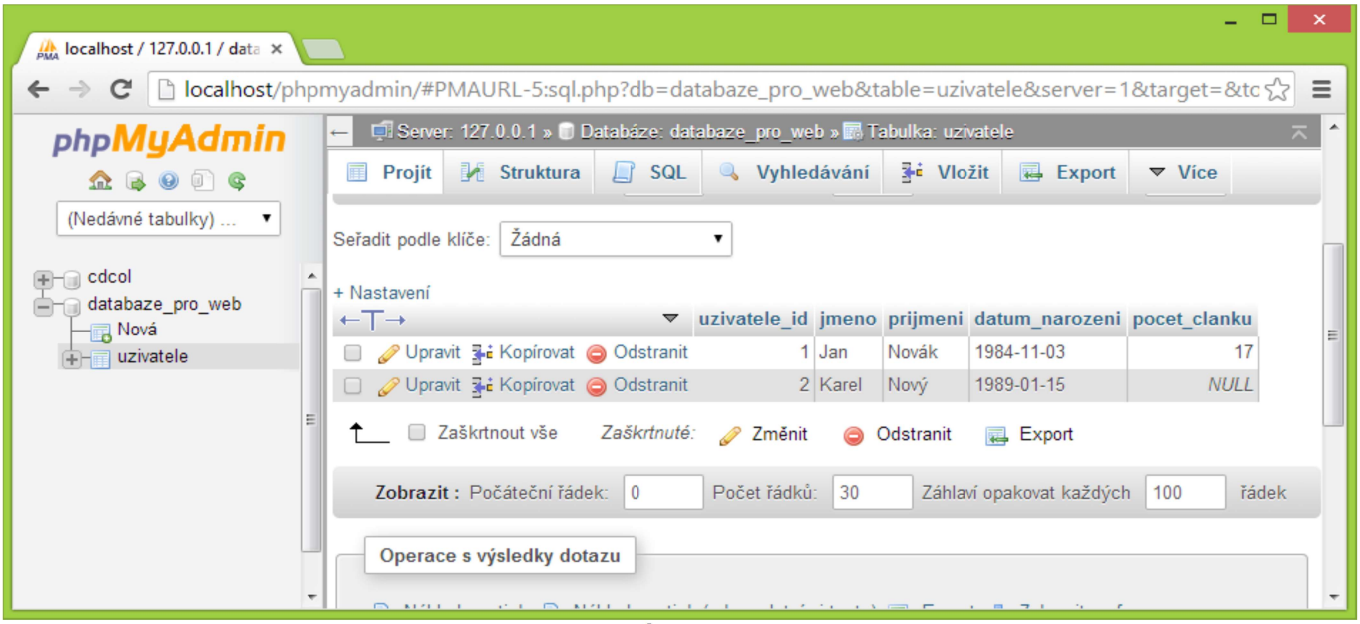
Pro jistotu ještě jednou ta samá část kódu tak, jak se správně:

```
Db::query('
    INSERT INTO uzivatele (jmeno, prijmeni, datum_narozeni)
    VALUES (?, ?, ?)
', $_POST['jmeno'], $_POST['prijmeni'], $datum);
```

Kód vyzkoušejme. Vložme nějakého uživatele:



A podívejme se do databáze, že v ní opravdu je:



Výpis dat

Do našeho jednoduchého skriptu přidejme ještě výpis dat z databáze do HTML tabulky. Tento PHP kód umístíme na konec dosavadního PHP bloku:

```
$uzivatele = Db::queryAll('
    SELECT *
    FROM uzivatele
');
echo('<h2>Uživatelé</h2><table border="1">');
foreach ($uzivatele as $u)
    {
    echo('<tr><td>' . htmlspecialchars($u['jmeno']));
    echo('</td><td>' . htmlspecialchars($u['prijmeni']));
    $datum = date("d.m.Y", strtotime($u['datum_narozeni']));
    echo('</td><td>' . htmlspecialchars($datum));
    echo('</td><td>' . htmlspecialchars($u['pocet_clanku']));
    echo('</td></tr>');
    }
echo('</table>');
```

Nejdůležitější je volání funkce Db::queryAll(). To vykoná databázový dotaz, stejně jako Db::query() a zároveň vrátí všechny řádky, které dotaz vybral. Budeme ji používat při čtení a Db::query() budeme používat při zápisu (přidání, editace, mazání). Samotný SQL dotaz obsahuje jen 4 slova. Dal by se přeložit jako "Vyber všechny sloupce z uživatelů". Právě hvězdička označuje všechny sloupce. Jelikož neupřesňujeme kteří uživatelé nás zajímají, vybere dotaz všechny řádky z tabulky.
Výsledkem dotazu je pole řádků, které si uložíme do proměnné $uzivatele. Následně pole proiterujeme pomocí foreach cyklu řádek po řádku a pro každý vyechujeme sloupeček do HTML tabulky. Nezapomeneme používat funkci htmlspecialchars(), jinak by si někdo mohl do jména vložit JavaScript a ten by se poté při výpisu jména spustil. Tomuto útoku se říká XSS.
Podívejme se na výsledek aplikace:

Můžete si zkusit přidávat uživatele, budou se objevovat v tabulce. To by bylo pro dnešní díl vše. Doufám, že se mi podařilo prolomit ledy a že jste úspěšně vytvořili svou první databázovou aplikaci. Příště začneme pracovat na slíbeném redakčním systému. Zdrojové kódy dnešní aplikace jsou jako vždy ke stažení v příloze.

4. díl - Programujeme neobjektový redakční systém v PHP (NERS)

V minulém dílu našeho seriálu tutoriálů o databázích v PHP pro úplné začátečníky jsme dokončili jednoduchou aplikaci k registraci a výpisu uživatelů. Úplné základy práce s databází tedy již ovládáme a nic nám nebrání k tomu, abychom si naprogramovali plnohodnotný redakční systém v PHP.

NERS

V několika málo dílech vytvoříme systém, pomocí kterého se budeme moci na náš web přihlašovat a nahrávat na něj články pomocí editoru. Systém jsem pojmenoval NERS (NEobjektový Redakční Systém). Má tyto atributy:

- Celý systém je napsaný **jen základními PHP funkcemi**, tedy bez objektů a bez šablon. Jedinou výjimkou je databázový wrapper, bez kterého se bohužel neobejdeme.
- Celý systém je obsažen **pouze v 6ti PHP souborech**
- Délka žádného souboru systému **nepřesahuje 100 řádků**. (asi o 5 řádků je delší jen editor.php, kde jsem nechal nějaké mezery kvůli přehlednosti).
- Systém stihneme **vytvořit velmi rychle** a to pouze **ve čtyřech tutoriálech**
- Systém obsahuje **uživatelské role, registraci, přihlašování, výpis článků a interaktivní editor článků**

Systém byl napsán tak, aby ho mohl **používat a rozšiřovat kdokoli bez ohledu na jeho znalosti PHP**. Bez objektů, knihoven a všeho dalšího se obešel díky kompromisům, zjednodušením a minimálním požadavkům. Systém **není určený pro vážné projekty** a**neukazuje, jak se v PHP správně programují informační systémy**. Ukazuje, jak lze **co nejjednodušeji dosáhnout požadovaného výsledku**. Pro **vážné informační systémy musíte použít MVC framework**. Na ITnetworku je na toto téma spoustu pokročilých seriálů a materiálů.

Příprava projektu

Než začneme programovat PHP skripty, připravíme si layout stránky a databázovou strukturu.

Layout

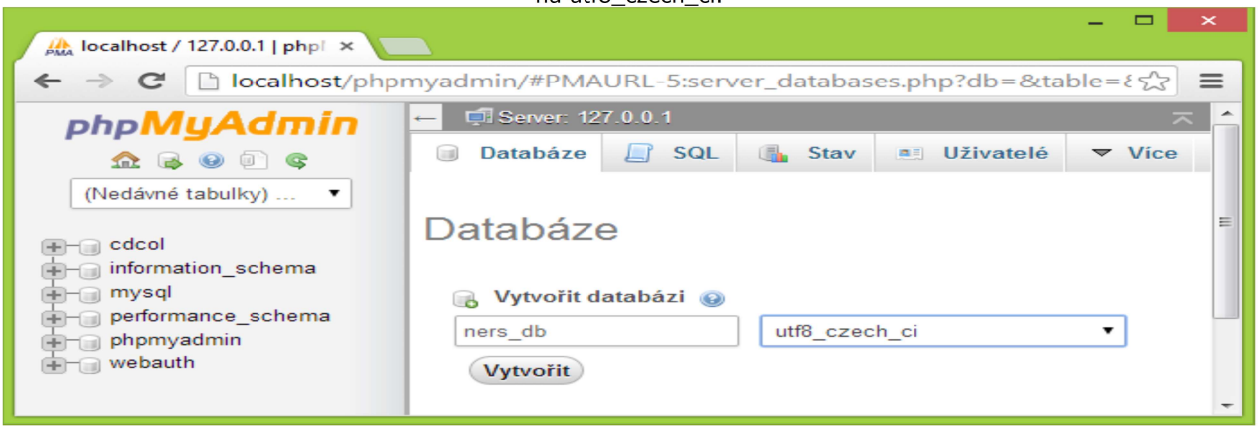
Nejprve budeme potřebovat nějaký layout (HTML kostru), do kterého doprogramujeme pomocí PHP požadované funkce. Určitě ho nebudeme tvořit od znovu, ale vypůjčíme si ho z místního ukázkového webu od Honzy Bittnera. V seriálu Webové stránky krok za krokem si otevřete poslední díl a kompletní web si stáhněte a prohlédněte. Pokud čemukoli nerozumíte, v seriálu naleznete detailní vysvětlení každé části. Web je jen čisté HTML, takže byste s ním neměli mít problém.

Založte si nový PHP projekt a přesuňte do něj obsah archivu s HoBiho webem. Všechny HTML stránky z něj odstraňte, nebudeme je potřebovat.

Databáze

Do projektu si vložte náš wrapper Db.php (ke stažení v 2. dílu tohoto seriálu). Dělal jsem v něm nějaké změny, pokud jste ho stahovali před 21.3.2014, stáhněte si ho prosím znovu.

Přejděte do PHPMyAdmin a vytvořte si nějakou novou databázi, já si ji pojmenoval ners_db. Nezapomeňte nastavit porovnávání na utf8_czech_ci.



Databázi si otevřete. Nyní si vytvoříme tabulky, budou jen dvě.

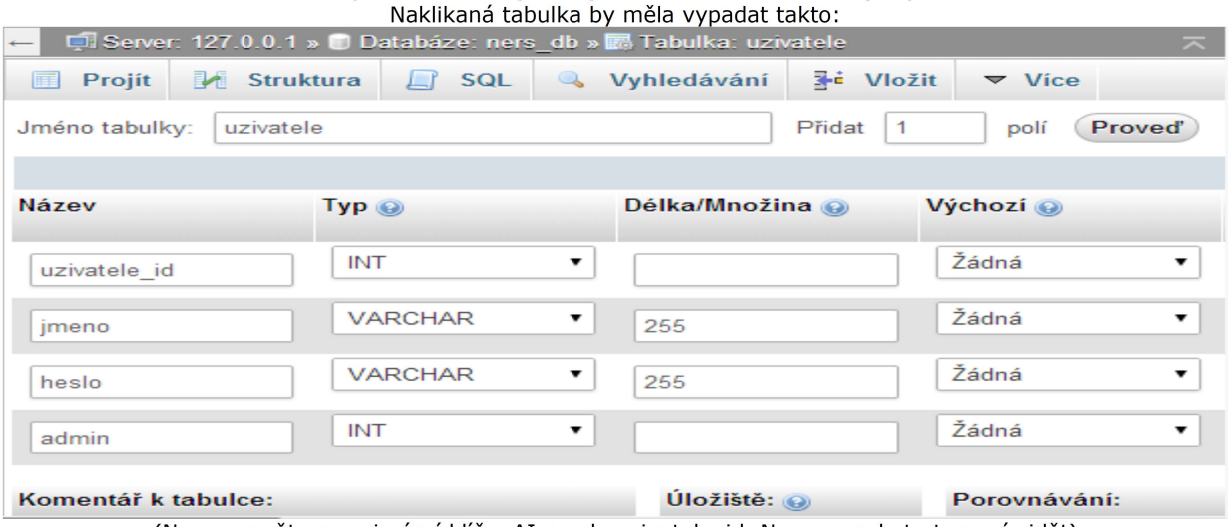
Uživatelé

Vytvořte si novou tabulku **uzivatele**. Bude mít celkem 4 sloupce (pole):

- **uzivatele_id** - typ int - primární klíč, autoincrement
- **jmeno** - typ varchar - délka 255
- **heslo** - typ varcha - délka 255
- **admin** - typ int

Sloupec admin určuje zda je uživatel administrátor (0/1).

Naklikaná tabulka by měla vypadat takto:



(Nezapomeňte na primární klíč a AI u pole uzivatele_id. Na screenshotu to není vidět)

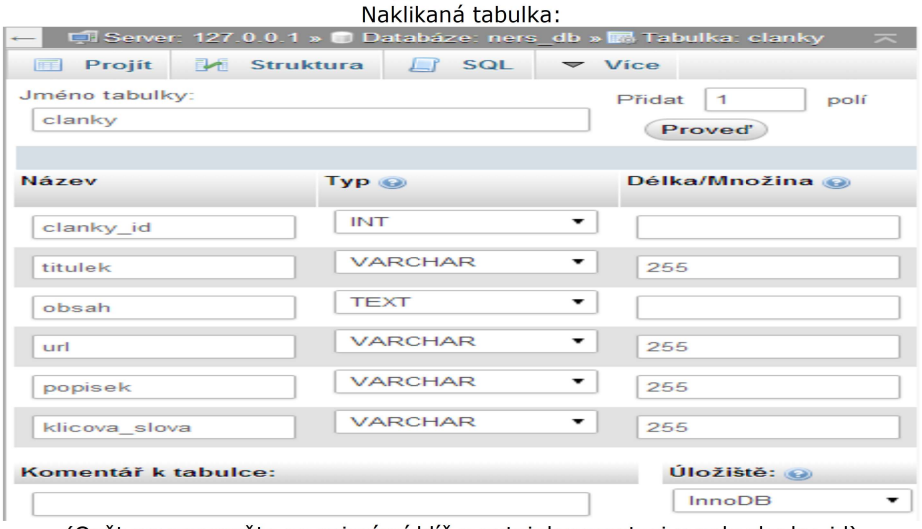
Uložíme tlačítkem Uložit níže.

Články

Přidáme ještě tabulku **clanky**. Ta bude obsahovat 6 sloupců (polí).

- **clanky_id** - typ int - primární klíč, autoincreament
- **titulek** - typ varchar - délka 255
- **obsah** - typ text
- **url** - varchar - délka 255
- **popisek** - varchar - délka 255
- **klicova_slova** - varchar - délka 255

Naklikaná tabulka:



(Opět nezapomeňte na primární klíč a autoinkrementaci u pole clanky_id)

Sloupec URL je url adresa, přes kterou budeme k článku přistupovat. Např. pro článek s titulkem "Úvodní článek" by URL mohlo vypsat jako "uvodni-clanek". Některé systémy články zobrazují jen podle číselného ID, což je však nepříjemné jak pro uživatele, tak pro vyhledávače. Pro text článku zvolíme datový typ text, varchar je vhodný zejména pro kratší texty.

Potvrdíme tlačítkem Uložit. Nyní máme vše připravené pro to, abychom se pustili do samotných PHP skriptů.

PHP Skripty

PHP skriptů bude tedy celkem 6. **Každý skript bude rozdělen na 2 části**. V první polovině bude **PHP blok s obsluhou**, v druhé polovině bude **HTML kód se stránkou a** nějakou příměsí PHP kódu s **výpisem výstupu obsluhy**. Tyto 2 části se v aplikaci většinou striktně oddělují do samostatných souborů, ale to pro naše účely zanedbáme. Jak jsem již uváděl, na prvním místě tohoto projektu je jednoduchost, díky které bude návrh aplikace trochu trpět. Malý projekt to zvládne, s něčím větším by to byl již problém.

Jako první se musíme do našeho systému zaregistrovat. Proto do kořenové složky s projektem přidáme soubor registrace.php

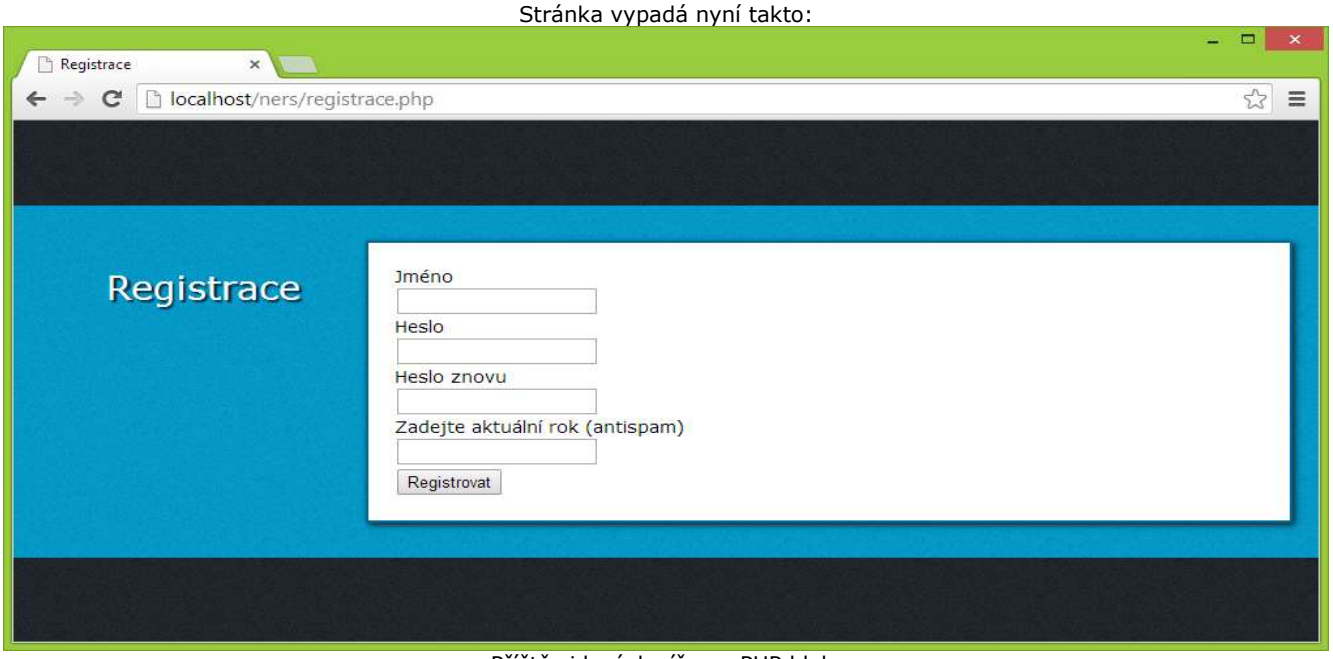
Do souboru vložíme nejprve následující HTML kód:

```
<!DOCTYPE html>
<html lang="cs-cz">
<head>
<meta charset="utf-8" />
<link rel="stylesheet" href="styl.css" type="text/css" />
```

```
<title>Registrace</title>
</head>

<body>
<article>
<div id="centrovac">
<header>
<h1>Registrace</h1>
</header>
<section>
<?php
if (isset($zprava))
echo('<p>' . $zprava . '</p>');
?>

<form method="post">
Jméno<br />
<input type="text" name="jmeno" /><br />
Heslo<br />
<input type="password" name="heslo" /><br />
Heslo znovu<br />
<input type="password" name="heslo_znovu" /><br />
Zadejte aktuální rok (antispam)<br />
<input type="text" name="rok" /><br />
<input type="submit" value="Registrovat" />
</form>
</section>
<div class="cistic"></div>
</div>
</article>
</body>
</html>
```

Je krátký a jednoduchý. Obsahuje HTML strukturu a v ní formulář pro registraci nového uživatele do systému. Uživatel zadá jméno, 2x heslo (pro kontrolu) a zadá aktuální rok, což je jednoduchá ochrana proti spamu. V HTML kódu je malá přímě PHP, která vypisuje text v proměnné $zprava, pokud tato proměnná existuje. Proměnnou budeme používat pro přenos zprávy z PHP bloku do HTML bloku.

Stránka vypadá nyní takto:



Příště si k ní dopíšeme PHP blok.

5. díl - NERS - Registrace uživatelů v PHP

V minulém dílu našeho seriálu tutoriálů o databázích v PHP pro úplné začátečníky jsme si připravili projekt NERS - NEobjektový Redakční Systém v PHP a rozpracovali jsme skript registrace.php.

Nyní nad HTML blok vložíme blok s PHP obsluhou:

```
<?php
session_start();
require('Db.php');
Db::connect('127.0.0.1', 'ners_db', 'root', '');

if ($_POST)
{
if ($_POST['rok'] != date('Y'))
$zprava = 'Chybně vyplněný antispam.';
else if ($_POST['heslo'] != $_POST['heslo_znovu'])
```

```
            $zprava = 'Hesla nesouhlasí';
        else
            {
        $existuje = Db::querySingle('
            SELECT COUNT(*)
            FROM uzivatele
            WHERE jmeno=?
            LIMIT 1
        ', $_POST['jmeno']);
        if ($existuje)
    $zprava = 'Uživatel s touto přezdívkou je již v databázi obsažen.';
        else
            {
            Db::query('
            INSERT INTO uzivatele (jmeno, heslo)
            VALUES (?, SHA1(?))
        ', $_POST['jmeno'], $_POST['heslo'] . "t&#ssdf54gh");
            $_SESSION['uzivatel_id'] = Db::getLastId();
            $_SESSION['uzivatel_jmeno'] = $_POST['jmeno'];
            $_SESSION['uzivatel_admin'] = 0;
            header('Location: administrace.php');
            exit();
            }
        }
    }
?>
```

Jako první voláme session_start(). Funkce nám umožní používat tzv. session (česky relace nebo někdy i sezení), která si pamatuje data uživatele, se kterým komunikujeme. **Tento řádek musí být na úplném začátku PHP souboru (ne bloku, opravdu souboru), ve kterém uživatelskou relaci používáme**. Před session_start() se nesmí nalézat žádné HTML, ani prázdné řádky, ani mezery, jinak nebude fungovat. To samé platí pro funkci header().

Dále načteme Db wrapper a připojíme se k databázi. Údaje jsou pro localhost, vy již víte, že na produkci vám je sdělí webhosting.

Další kód je obsluha HTML formuláře. Pokud zadaný rok nesouhlasí s aktuálním, uložíme si chybovou zprávu. Jinak pokračujeme dále a stejným způsobem ověříme shodu obou zadaných hesel. Dostáváme se k samotné registraci.

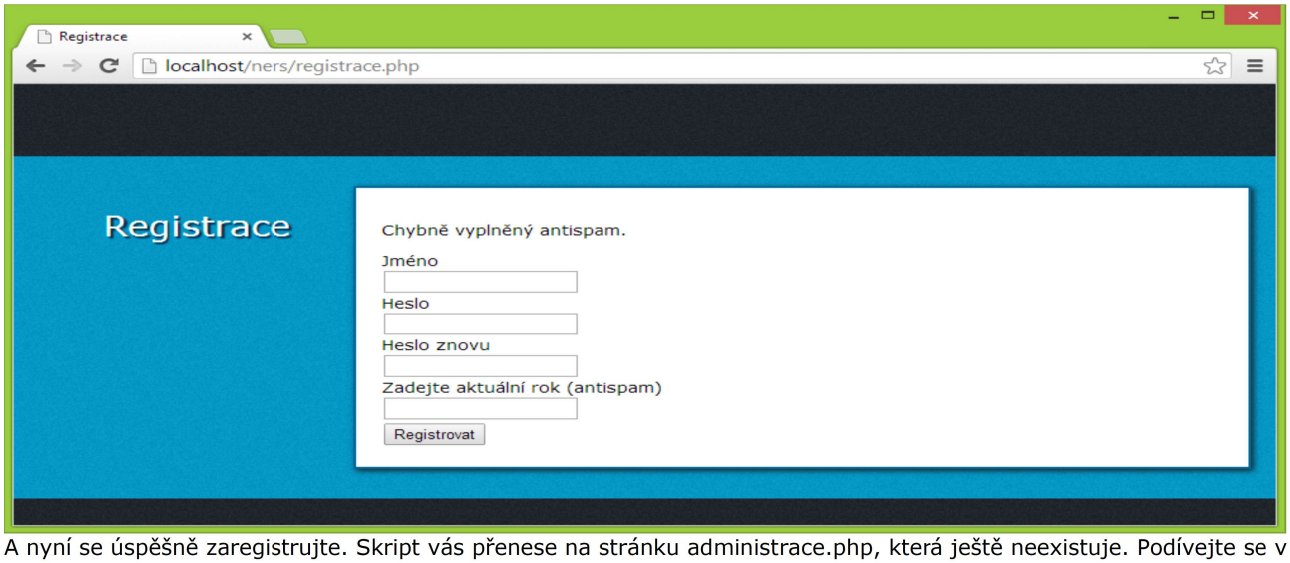
Nejprve necháme databázi spočítat kolik je v ní uživatelů se zadaným jménem. Slouží k tomu SQL klauzule SELECT COUNT(*). Hvězdička označuje, že do výpočtu zahrnujeme všechny sloupce. Abychom databázi zbytečně zatěžovali, dáme na počet položek ještě LIMIT 1. Stačí nám, když zjistíme, že je v databázi nějaký uživatel s tímto jménem a dále se již databáze snažit nemusí. Dotaz zavoláme funkcí Db::querySingle(). Ta se používá v případě, když z databáze čteme pouze jednu hodnotu jednoho řádku. My zde čteme pouze jedno číslo - počet uživatelů s tímto jménem. Pokud jsme někoho našli, jméno je již obsazené a uložíme si chybovou hlášku.

SQL kód pro vložení uživatele do databáze je velmi podobný tomu z minulých lekcí. O něco složitější je zde ukládání hesla. Nejprve heslo uživatele vylepšíme a to přidáním tzv. soli (salt). To je úplně náhodný řetězec znaků, který se k heslu připojí kvůli bezpečnosti, nějaký si vymyslete. Uživatelé totiž rádi zadávají např. "heslo", "password", "123456" a podobně. Takto zadaná hesla jsou velmi nebezpečná a lze je jednoduše odhadnout. Po "osolení" z nich vznikne: "heslot&#ssdf54gh", "passwordt&#ssdf54gh", "123456t&#ssdf54gh", což prakticky znemožní zpětné uhodnutí onoho hesla. Osolené heslo nevložíme do databáze přímo a to proto, že databázi nám může někdo ukradnout a potom by viděl všechna hesla všech uživatelů.

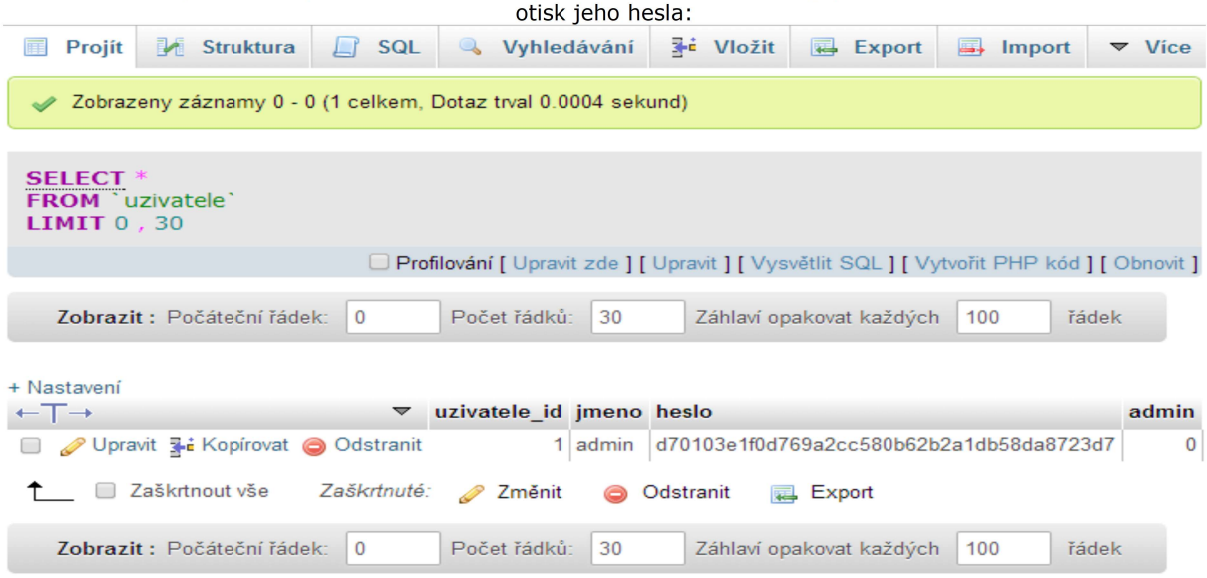
Heslo uložíme pomocí SQL funkce SHA1(). To je tzv. hashovací funkce, která z daného hesla vypočítá otisk. Nejbezpečnější způsob uchování hesla v databázi je totiž heslo vůbec neuchovat, ale uchovat si pouze jeho otisk. Otisk je dlouhý řetězec zdánlivě nesouvisejících znaků a číslic a nelze z něj původní heslo zjistit. Hashovací funkce je tedy pouze jednosměrná. Když budeme chtít zjistit, zda uživatel zadal správné heslo, jednoduše ze zadaného hesla vypočítáme otisk stejným způsobem a tento otisk porovnáme s uloženým otiskem v databázi. To vše uvidíte až budeme programovat přihlašování.

Po vložení uživatele do databáze si vytvoříme uživatelskou relaci. V PHP je podobně jako $_GET nebo $_POST také superglobální pole $_SESSION. Do tohoto pole si můžeme ukládat data, které souvisí s tím uživatelem, se kterým právě komunikujeme. Relace vyprší ve výchozím nastavení PHP za 24 minut nečinnosti nebo po zavření prohlížeče. Jakmile ji jednou vytvoříme v nějakém skriptu, zůstanou nám její data přístupná i pro ostatní PHP skripty v aplikaci. My zde uživatele po registraci rovnou i přihlásíme a to tak, že do relace uložíme jeho id, jméno a zda je administrátor. Id (primární klíč) záznamu naposledy vloženého do databáze získáme pomocí funkce Db::getLastId(). Nakonec se přesměrujeme funkcí header() na stránku administrace.php a skript zastavíme. Pokud se něco nepodařilo, skript bude pokračovat a vypíše chybovou hlášku.

Zkusme si udělat nějakou chybu:



A nyní se úspěšně zaregistrujte. Skript vás přenese na stránku administrace.php, která ještě neexistuje. Podívejte se v phpMyAdmin do tabulky uzivatele (otevřete tabulku a vyberte z menu nahoře Projít). Vidíme zde nově vloženého uživatele a otisk jeho hesla:



Jako heslo jsem zadal "admin", výše vidíte otisk, co vrátila funkce SHA1().

Administrace

Vytvořme si nyní skript administrace.php. Bude to takový rozcestník, na který aplikace uživatele přenese po zaregistrování nebo po přihlášení. Jeho HMTL část bude následující:

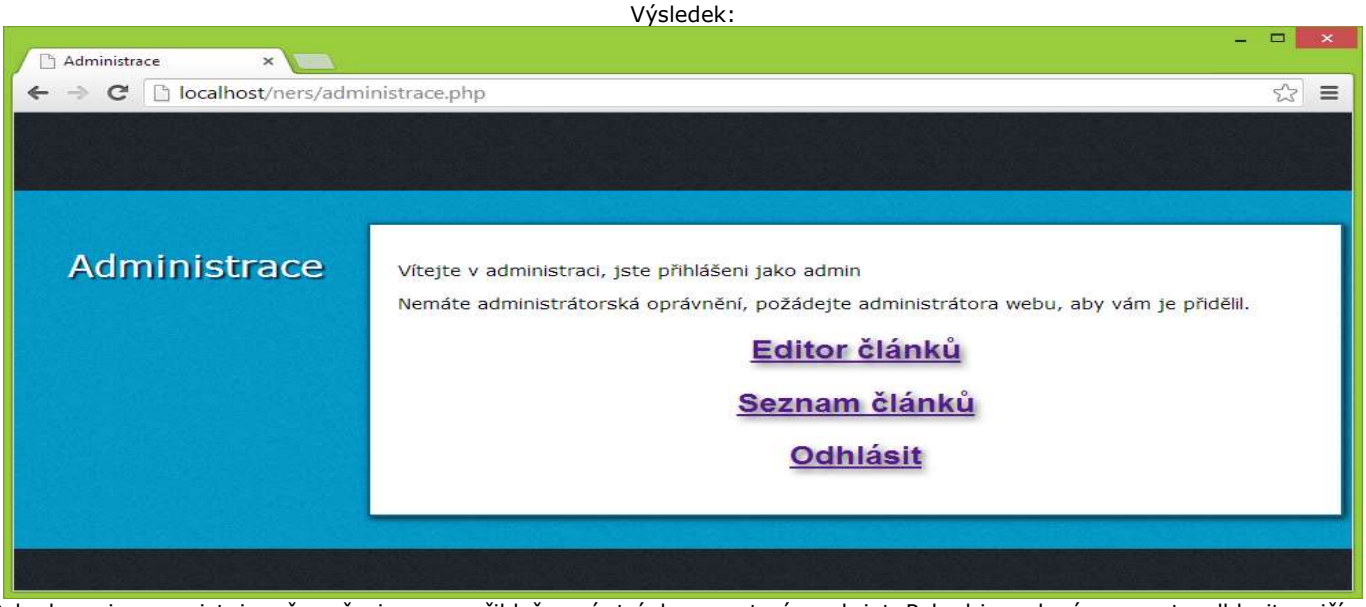
```
<!DOCTYPE html>
<html lang="cs-cz">
<head>
<meta charset="utf-8" />
<link rel="stylesheet" href="styl.css" type="text/css" />
<title>Administrace</title>
</head>

<body>
<article>
<div id="centrovac">
<header>
<h1>Administrace</h1>
</header>
<section>
<p>Vítejte v administraci, jste přihlášeni jako <?= htmlspecialchars($_SESSION['uzivatel_jmeno']) ?></p>
<?php
if (!$_SESSION['uzivatel_admin'])
echo('Nemáte administrátorská oprávnění, požádejte administrátora webu, aby vám je přidělil.');
?>
<h2><a href="editor.php">Editor článků</a></h2>
<h2><a href="clanky.php">Seznam článků</a></h2>
<h2><a href="administrace.php?odhlasit">Odhlásit</a></h2>
</section>
<div class="cistic"></div>
</div>
</article>
```

```
</body>
</html>
```

Kód obsahuje pouze 3 zajímavé věci
Vypisujeme zde jméno přihlášeného uživatele z relace (session).
Pokud není přihlášený uživatel administrátor, informujeme ho o tom.
Pro odhlášení uživatele odkážeme na skript administrace.php s GET parametrem odhlasit.
Nad kód vložíme jednoduchý PHP blok

```
<?php
session_start();
if (!isset($_SESSION['uzivatel_id']))
{
header('Location: prihlaseni.php');
exit();
}

if (isset($_GET['odhlasit']))
{
session_destroy();
header('Location: prihlaseni.php');
exit();
}
?>
```

Výsledek:



Pokud session neexistuje, přesměrujeme na přihlašovací stránku a zastavíme skript. Pokud je zadaný parametr odhlasit, zničíme session pomocí PHP funkce session_destroy() a opět přesměrujeme na přihlášení.

6. díl - NERS - Editor článků v PHP

V minulém dílu našeho seriálu tutoriálů o databázích v PHP pro úplné začátečníky jsme dokončili administrační stránku jednoduchého redakčního systému. V dnešním dílu přidáme přihlašování.

Přihlašování

Vytvoříme skript prihlaseni.php. Jeho HTML část bude následující:

```
<!DOCTYPE html>
<html lang="cs-cz">
<head>
<meta charset="utf-8" />
<link rel="stylesheet" href="styl.css" type="text/css" />
<title>Přihlášení do administrace</title>
</head>

<body>
<article>
<div id="centrovac">
<header>
<h1>Přihlášení do administrace</h1>
</header>
<section>
<?php
if (isset($zprava))
echo('<p>' . $zprava . '</p>');
?>

<form method="post">
Jméno<br />
<input type="text" name="jmeno" /><br />
Heslo<br />
```

```
<input type="password" name="heslo" /><br />
<input type="submit" value="Přihlásit" />
</form>

<p>Pokud ještě nemáte účet, <a href="registrace.php">zaregistrujte se</a>.</p>
</section>
<div class="cistic"></div>
</div>
</article>
</body>
</html>
```

Vidíme PHP direktivu pro výpis chybové zprávy (pokud existuje) a formulář se jménem a heslem. Kód je téměř totožný s tím u registračního formuláře a měl by být tedy srozumitelný.

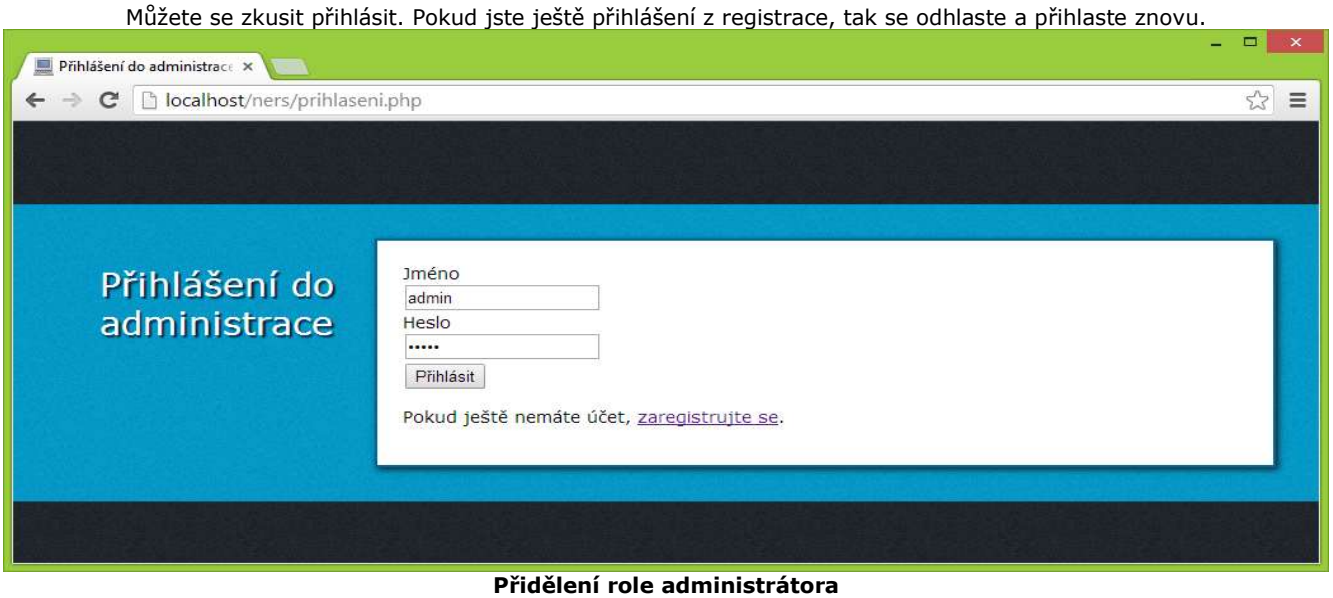
Nad HTML vložíme následující PHP blok:

```
<?php
session_start();
require('Db.php');
Db::connect('127.0.0.1', 'ners_db', 'root', '');

if (isset($_SESSION['uzivatel_id']))
{
header('Location: administrace.php');
exit();
}

if ($_POST)
{
$uzivatel = Db::queryOne('
SELECT uzivatele_id, admin
FROM uzivatele
WHERE jmeno=? AND heslo=SHA1(?)
', $_POST['jmeno'], $_POST['heslo'] . "t&#ssdf54gh");
if (!$uzivatel)
$zprava = 'Neplatné uživatelské jméno nebo heslo';
else
{
$_SESSION['uzivatel_id'] = $uzivatel['uzivatele_id'];
$_SESSION['uzivatel_jmeno'] = $_POST['jmeno'];
$_SESSION['uzivatel_admin'] = $uzivatel['admin'];
header('Location: administrace.php');
exit();
}
}
?>
```

První řádky nám zpřístupní session a připojí se k databázi. Pokud je uživatel přihlášený, tak mu přihlašovací stránku zobrazovat nebudeme a přesměrujeme ho rovnou na administraci a skript zastavíme.

Pokud byl odeslán formulář, osolíme zadané heslo stejnou solí, jako při registraci a uděláme z něj opět stejný otisk pomocí SQL funkce SHA1(). Uživatele se zadaným jménem a otiskem se následně pokusíme najít. Pokud ho nenajdeme, uložíme chybovou hlášku. V případě úspěchu uložíme načtené informace o uživateli do session a tím ho přihlásíme. Dále ho přesměrujeme na administraci.

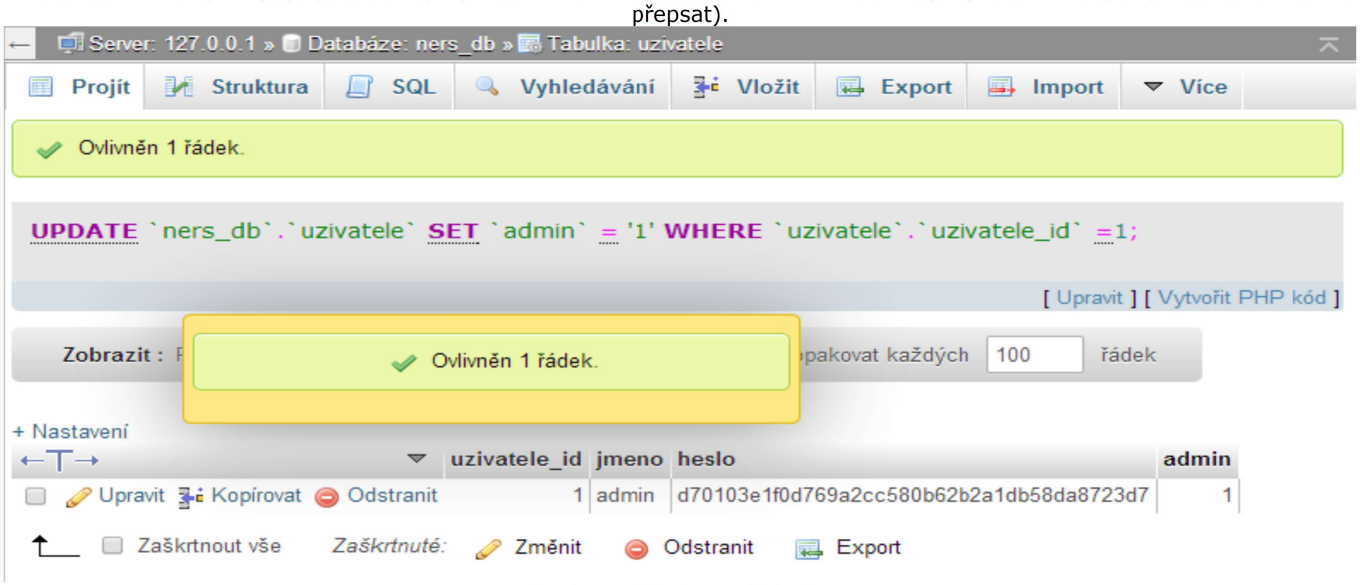
Můžete se zkusit přihlásit. Pokud jste ještě přihlášení z registrace, tak se odhlaste a přihlaste znovu.



Přidělení role administrátora

V našem systému máme u každého uživatele sloupec admin. Ten bude nabývat hodnoty 0 nebo 1 podle toho, zda je uživatel administrátor. Tuto hodnotu může nastavit nově registrovanému uživateli pouze administrátor webu pomocí phpMyAdmin.

Přesuneme se tedy do phpMyAdmin a přepneme hodnotu admin u našeho účtu na 1 (na políčko stačí jen poklikat a hodnotu přepsat).



Nyní se odhlaste a přihlaste.

Editor článků

Přejděme k editoru článků. Vytvořme soubor editor.php a vložme do něj HTML blok:

```
<!DOCTYPE html>
<html lang="cs-cz">
<head>
<meta charset="utf-8" />
<link rel="stylesheet" href="styl.css" type="text/css" />
<title>Editor článků</title>
</head>

<body>
<article>
<div id="centrovac">
<header>
<h1>Editor článků</h1>
</header>
<section>
<?php
if (isset($zprava))
echo('<p>' . $zprava . '</p>');
?>

<form method="post">
<input type="hidden" name="clanky_id" value="<?= htmlspecialchars($clanek['clanky_id']) ?>" /><br />
Titulek<br />
<input type="text" name="titulek" value="<?= htmlspecialchars($clanek['titulek']) ?>" /><br />
URL<br />
<input type="text" name="url" value="<?= htmlspecialchars($clanek['url']) ?>" /><br />
Popisek<br />
<input type="text" name="popisek" value="<?= htmlspecialchars($clanek['popisek']) ?>" /><br />
Klíčová slova<br />
<input type="text" name="klicova_slova" value="<?= htmlspecialchars($clanek['klicova_slova']) ?>" /><br />
<textarea name="obsah"><?= htmlspecialchars($clanek['obsah']) ?></textarea>
<input type="submit" value="Odeslat" />
</form>
</section>
<div class="cistic"></div>
</div>
</article>
<script type="text/javascript" src="//tinymce.cachefly.net/4.0/tinymce.min.js"></script>
<script type="text/javascript">
tinymce.init({
selector: "textarea[name=obsah]",
plugins: [
"advlist autolink lists link image charmap print preview anchor",
"searchreplace visualblocks code fullscreen",
"insertdatetime media table contextmenu paste"
],
toolbar: "insertfile undo redo | styleselect | bold italic | alignleft aligncenter alignright alignjustify | bullist numlist outdent indent | link image",
entities: "160,nbsp",
```

```
entity_encoding: "named",
entity_encoding: "raw"
});
</script>
</body>
</html>
```

Kód obsahuje opět jen jednoduchý HTML formulář a výpis chybové zprávy. Data se do formulářových polí vypisují z pole $clanek. Formulář má ještě jednu zvláštnost a tou je skryté pole s ID daného článku. Podle tohoto pole poznáme, zda vkládáme nový článek (bude prázdné) nebo zda editujeme existující (bude v něm ID tohoto článku).

Zajímavý je JavaScript na konci souboru, ve které načteme WYSIWYG editor TinyMCE. To je editor vzhledově podobný např. MS Wordu, který nám generuje HTML kód podle toho, co v něm naklikáme.

První skript je odkaz na online úložiště, ze kterého si prohlížeč TinyMCE stáhne. Druhý skript obsahuje nastavení editoru, zvolíme, že editor chceme vytvořit z textarea s názvem obsah. Další řádky nastavují pluginy a vypínají převod české diakritiky na entity, což je jinak poměrně nepříjemná záležitost.

Nad HTML kód dodejme obslužný PHP blok:

```
<?php
session_start();
if (empty($_SESSION['uzivatel_admin']))
die('Nedostatecna opravneni');

require('Db.php');
Db::connect('127.0.0.1', 'ners_db', 'root', '');

$clanek = array(
'clanky_id' => '',
'titulek' => '',
'obsah' => '',
'url' => '',
'popisek' => '',
'klicova_slova' => '',
);
if ($_POST)
{
if (!$_POST['clanky_id'])
{
Db::query('
INSERT INTO clanky (titulek, obsah, url, popisek, klicova_slova)
VALUES (?, ?, ?, ?, ?)
', $_POST['titulek'], $_POST['obsah'], $_POST['url'], $_POST['popisek'], $_POST['klicova_slova']);
}
else
{
Db::query('
UPDATE clanky
SET titulek=?, obsah=?, url=?, popisek=?, klicova_slova=?
WHERE clanky_id=?
', $_POST['titulek'], $_POST['obsah'], $_POST['url'], $_POST['popisek'], $_POST['klicova_slova'], $_POST['clanky_id']);
}
header('Location: index.php?clanek=' . $_POST['url']);
exit();
}
else if (isset($_GET['url']))
{
$nactenyClanek = Db::queryOne('
SELECT *
FROM clanky
WHERE url=?
', $_GET['url']);
if ($nactenyClanek)
$clanek = $nactenyClanek;
else
$zprava = 'Článek nebyl nalezen';
}

?>
```

Pokud uživatel není administrátor, zastavíme celý skript s chybovou hláškou. Pro kontrolu přihlášení administrátora nám nestačí isset(), ale musíme použít empty(). V session totiž může existovat klíč 'uzivatel_admin', ale může mít hodnotu 0.

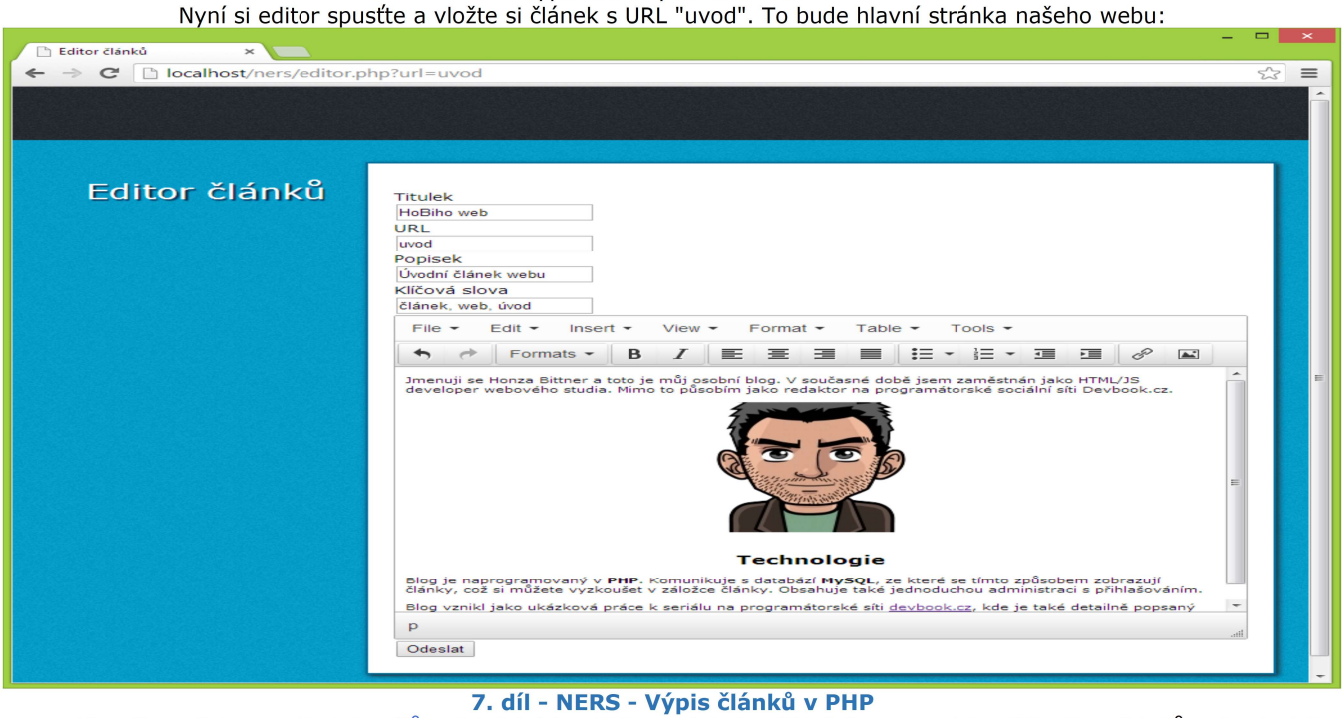
Dále se připojíme k databázi a do proměnné $clanek si připravíme pole s prázdnými hodnotami. To aby se do formuláře nic nevypsalo a zároveň nám PHP neohlásilo chybu s neexistující proměnnou.

Pokud byl odeslán formulář, podíváme se do skrytého pole. Pokud je prázdné, vložíme nový článek do databáze. Pokud je v skrytém poli nějaká hodnota, updatujeme článek s tímto ID. SQL příkaz UPDATE jsme si ještě nepředstavovali, ale je velmi jednoduchý. Pomocí SET jednoduše nastavíme pole která potřebujeme. Co je důležité je nezapomenout na klauzuli WHERE, kde určíme které řádky se mají takto updatovat. Bez ní by se updatovaly všechny články na tyto hodnoty!

Po přidání nebo editaci článku na něj přesměrujeme.

Pokud nebyl odeslaný formulář, podíváme se, zda nemáme v GET url článku. To by znamenalo, že chceme nějaký editovat a proto se do pole $clanek pokusíme nahrát data z článku s tímto URL. Data se potom předvyplní do polí formuláře. Při neúspěchu vypíšeme chybovou hlášku.

Nyní si editor spusťte a vložte si článek s URL "uvod". To bude hlavní stránka našeho webu:

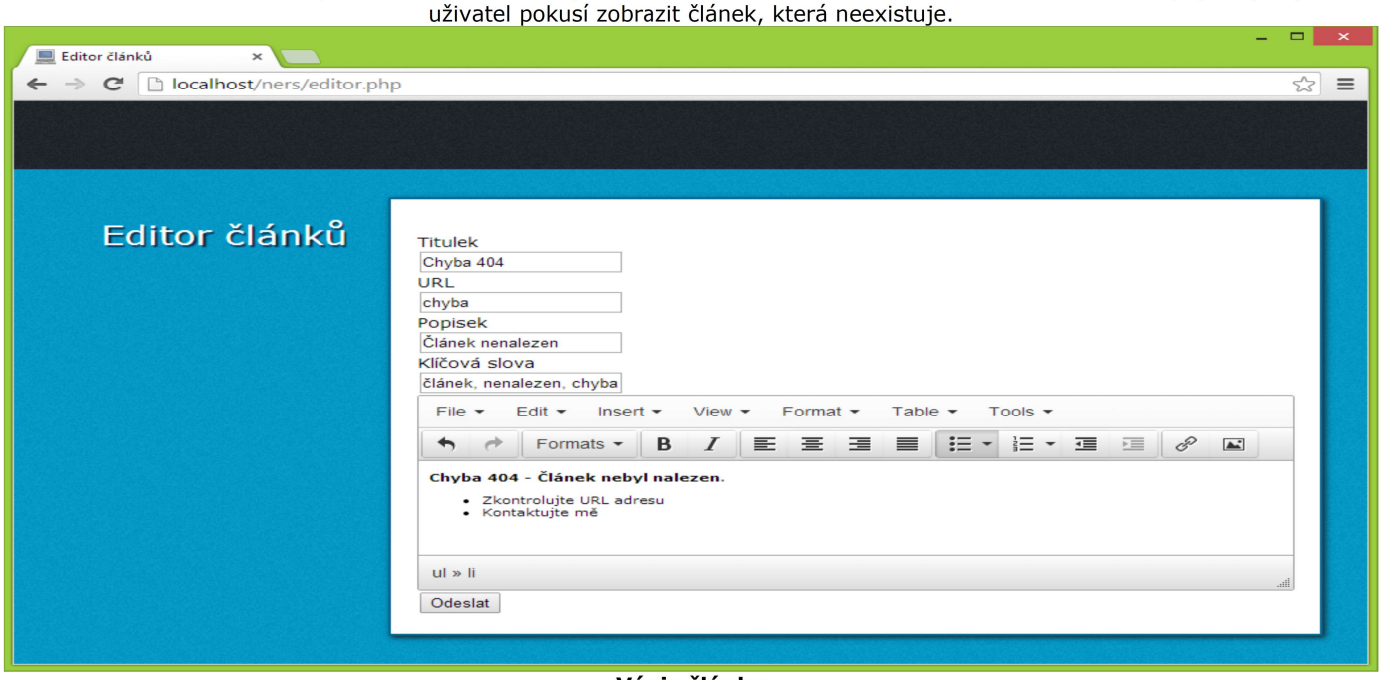


7. díl - NERS - Výpis článků v PHP

V minulém dílu našeho seriálu tutoriálů o databázích v PHP pro úplné začátečníky jsme dokončili editor článků jednoduchého redakčního systému a vytvořili úvodní stránku. V dnešním dílu vytvoříme indexovou stránku, která bude články zobrazovat.

Chybová stránka

Pomocí editoru článků si vytvořte článek s URL chyba a uložte ho do databáze. Tento článek se zobrazí v případě, když se uživatel pokusí zobrazit článek, která neexistuje.



Výpis článku

Pro výpis jednotlivých článků vytvoříme soubor index.php, do kterého vložíme následující HTML blok:

```
<!DOCTYPE html>
<html lang="cs-cz">

    <head>
        <meta charset="utf-8" />
        <meta name="description" content="<?= htmlspecialchars($clanek['popisek']) ?>" />
        <meta name="keywords" content="<?= htmlspecialchars($clanek['klicova_slova']) ?>" />
        <link rel="shortcut icon" href="obrazky/ikona.ico" />
        <link rel="stylesheet" href="styl.css" type="text/css" />
        <title><?= htmlspecialchars($clanek['titulek']) ?></title>
    </head>

    <body>
        <header>
```

```
<div id="logo"><h1>HoBi</h1></div>
<nav>
<ul>
<li><a href="index.php?clanek=uvod">Domů</a></li>
<li><a href="clanky.php">Články</a></li>
<li><a href="index.php?clanek=kontakt">Kontakt</a></li>
</ul>
</nav>
</header>

<article>
<div id="centrovac">
<header>
<h1><?= htmlspecialchars($clanek['titulek']) ?></h1>
</header>

<section>
<?= $clanek['obsah'] ?>
</section>
<div class="cistic"></div>
</div>
</article>

<footer>
Vytvořil &copy;HoBi 2013 pro <a href="http://devbook.cz">DEVBOOK.CZ</a>
<a href="administrace.php">Administrace</a>
</footer>
</body>
</html>
```

V kódu opět používáme proměnnou $clanek. Tentokrát z ní vypisujeme data do HTML hlavičky a obsah článku do těla stránky.
Jako vždy stránku doplníme ještě obslužným PHP blokem, který vložíme nad HTML blok:

```
<?php

require('Db.php');
Db::connect('127.0.0.1', 'ners_db', 'root', '');

if (isset($_GET['clanek']))
$url = $_GET['clanek'];
else
$url = 'uvod';

$clanek = Db::queryOne('
SELECT *
FROM clanky
WHERE url=?
', $url);
if (!$clanek)
{
if ($url != 'chyba')
{
header('Location: index.php?clanek=chyba');
exit();
}
else
die('Nebyl nalezen chybovy clanek');
}

?>
```

Kód je extrémně jednoduchý. Po připojení k databázi se podíváme do GETu, zda máme v adrese zadané url článku. Pokud ano, nastavíme proměnnou $url na tuto hodnotu. Pokud ne, nastavíme ji na úvodní stránku.
Podle URL se pokusíme načíst daný článek z databáze do proměnné $clanek. Pokud se to nepovedlo, přesměrujeme na článek s url chyba. Pokud nastala chyba na článku chyba (i to se může stát), ukončíme běh aplikace s chybovou hláškou.
To je vše. Přejdčme na index.php:

Vidíme, že se článek načetl z databáze a vypadá stejně, jako jsme ho napsali v editoru.

Výpis seznamu článků

Kromě jednoho článku by měl náš systém umět vypsat i seznam všech článků v databázi, seřazených od nejnovějších po nejstarší. Pokud je přihlášený administrátor, měl by mít možnost články v seznamu editovat a mazat.

Vytvoříme soubor clanky.php s následujícím HTML blokem:

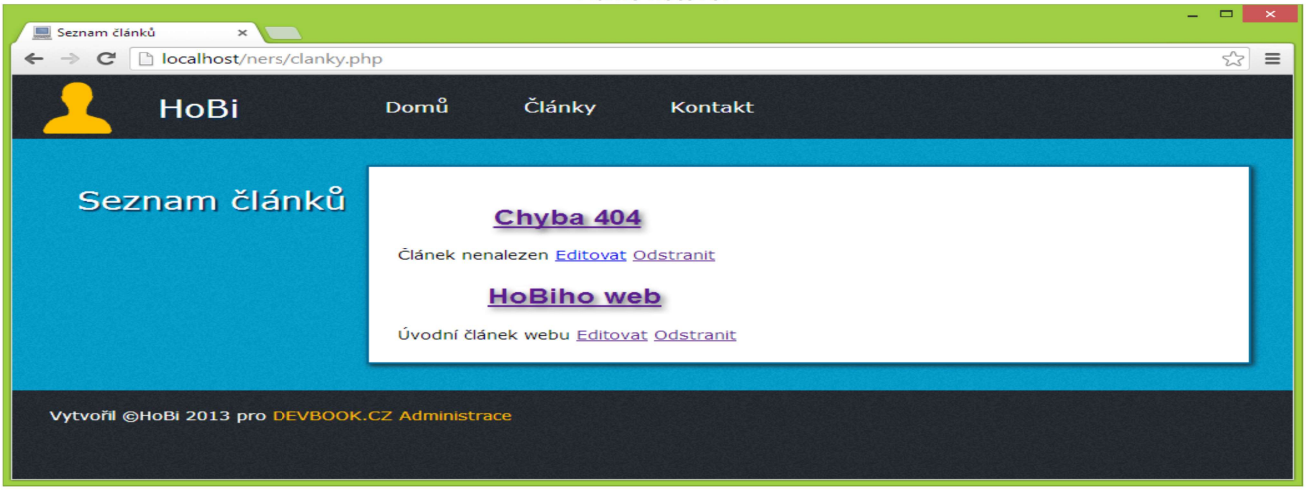
```
<!DOCTYPE html>
<html lang="cs-cz">

<head>
<meta charset="utf-8" />
<link rel="shortcut icon" href="obrazky/ikona.ico" />
<link rel="stylesheet" href="styl.css" type="text/css" />
<title>Seznam článků</title>
</head>

<body>
<header>
<div id="logo"><h1>HoBi</h1></div>
<nav>
<ul>
<li><a href="index.php?clanek=uvod">Domů</a></li>
<li><a href="clanky.php">Články</a></li>
<li><a href="index.php?clanek=kontakt">Kontakt</a></li>
</ul>
</nav>
</header>

<article>
<div id="centrovac">
<header>
<h1>Seznam článků</h1>
</header>

<section>
<table>
<?php
foreach ($clanky as $clanek)
{
echo('<tr><td><h2>
<a href="index.php?clanek=' . htmlspecialchars($clanek['url']) . '">
' . htmlspecialchars($clanek['titulek']) . '</a>
</h2>' . htmlspecialchars($clanek['popisek']));
if (!empty($_SESSION['uzivatel_admin']))
echo(' <a href="editor.php?url=' . htmlspecialchars($clanek['url']) . '">Editovat</a>
<a href="clanky.php?odstranit=' . htmlspecialchars($clanek['clanky_id']) . '">Odstranit</a>
');
echo('</td></tr>');
}
?>
</table>
```

```
</section>
<div class="cistic"></div>
</div>
</article>

<footer>
Vytvořil &copy;HoBi 2013 pro <a href="http://devbook.cz">DEVBOOK.CZ</a>
<a href="administrace.php">Administrace</a>
</footer>
</body>
</html>
```

Pracujeme zde s proměnnou $clanky, ve které jsou uložené jednotlivé články z databáze. Ty vypíšeme pod sebe do tabulky a pokud je přihlášený administrátor, přidáme k nim i odkazy na vymazání a editaci. Editaci samozřejmě zajišťuje skript editor.php, vymazání provede skript clanky.php.
Nad HTML dodejme náš poslední PHP blok:

```
<?php
session_start();

require('Db.php');
Db::connect('127.0.0.1', 'ners_db', 'root', '');

if (isset($_GET['odstranit']) && !empty($_SESSION['uzivatel_admin']))
{
Db::query('
DELETE FROM clanky
WHERE clanky_id=?
', $_GET['odstranit']);
header('Location: clanky.php');
exit();
}

$clanky = Db::queryAll('
SELECT *
FROM clanky
ORDER BY clanky_id DESC
');

?>
```

Pokud je v GET parametr "odstranit" a je přihlášený administrátor, vymažeme článek s ID, které nám přišlo v GETu. Do proměnné $clanky načteme všechny články pomocí jednoduchého SQL dotazu.
Máme hotovo.



Nyní můžete psát libovolné množství článků na svůj blog, web firmy, zápisník, zkrátka k čemu budete svůj systém používat
Ještě jednou zopakuji, že účelem tohoto seriálu bylo zejména prolomit ledy a umožnit vyzkoušet práci s databází úplně všem. Určitě se podívejte do sekce Objektově orientované programování v PHP, kde je krok za krokem vysvětleno jak se tvoří moderní informační systémy. Kompletní kód systému NERS je v příloze ke stažení.

8. díl - Začínáme s databázemi

Vítejte u dalšího dílu seriálu o tvorbě dynamických doplňků na webové stránky v PHP. Dnes se podíváme na zoubek databázím.

Co je to databáze

Databáze je úložiště dat na serveru. Naše skripty z něj můžou jak číst, tak do něj i zapisovat. Obslužné programy se automaticky starají o to, aby nedocházelo ke kolizím při současném přístupu více uživatelů. Kromě toho poskytují nejrůznější pomocné funkce pro práci s daty.
V tomto seriálu se zaměříme na databázi MySQL, která se v současné době používá asi nejčastěji. Zkratka SQL znamená "structured query language", což se obvykle překládá jako "strukturovaný dotazovací jazyk" a pomocí něj s databází komunikujeme (po HTML a PHP je to tedy už třetí jazyk, který je potřeba se naučit). Termínem "dotaz" (query) se rozumí jakýkoli příkaz pro databázi, ne jenom skutečné otázky jako třeba "kolik je v téhle tabulce řádků?". V dalším textu ale tohle slovo stejně používat nebudu, takže je to jedno.

MySQL (a všechny ostatní databáze používající jazyk SQL) je tzv. relační databáze. Data jsou v ní uspořádána v **tabulkách**. Řádky představují jednotlivé záznamy a sloupce pak datové položky uvnitř záznamů. Představit si to můžete třeba takhle:

SPZ	typ	majitel
LNC 2022	Škoda 120	Jiří Nebozíz
1U0 8729	Tatra 815	Sopwith Camel
1P2 3230	Škoda Felicia	Jarmila Nováková
PZA 4876	Fiat 500	Kuo Ču Nguyen

Řádky můžeme celkem jednoduše přidávat, mazat a přepisovat a jejich počet obvykle není nijak omezen. Strukturu tabulky, tedy počet a jména sloupců, si zvolíme jednou při vytváření tabulky a pak už s nimi většinou nehýbeme (teoreticky to sice jde, ale moc často se to nedělá).
Jinak než do tabulek data ukládat nejdou, takže i kdybychom potřebovali uložit třeba jenom jedno jediné číslo, musíme si na to vyrobit tabulku 1x1.
Slovo "relační" znamená, že mezi jednotlivými tabulkami můžou vznikat vztahy neboli relace. Pokud máme např. výše uvedenou tabulku automobilů a k ní třeba ještě tabulku lidí se jménem, adresou a datem narození, můžeme si mezi nimi představit vztah "člověk vlastní auto". Ale je to jenom pomyslné, mezi tabulkami nevedou žádné ukazatele nebo jiné fyzické vazby.

Práce s databází

Až na výjimky se dá databáze najít prakticky na každém serveru, včetně většiny freehostingů. Jediné, co potřebujete udělat, je databázi si nějak aktivovat a získat k ní přihlašovací údaje. Tohle je obecně na každém serveru jiné, proto vám nemůžu podrobně popsat každé kliknutí. Prostě zapátrejte v administračním rozhraní po slovech jako "databáze" nebo "MySQL" a postupujte podle pokynů, které tam najdete (jako příklady budu používat servery IC a Endora, které znám). Jakmile projdete touhle první fází, funguje všechno ostatní všude stejně.

Krok první: připojení k databázi

Každý skript, který chce pracovat s databází, se k ní nejdřív musí připojit. Když říkám každý, myslím tím opravdu úplně každý PHP soubor - nestačí to jenom jednou. Naštěstí známe příkazy include a require, takže připojovací kód nemusíme pokaždé opisovat. Funkce pro připojení vypadá takhle:

```
mysql_connect(jméno hostitele, přihlašovací jméno, heslo)
```

- Jméno hostitele se dozvíte při aktivaci databáze. Např. na IC je to 'mysql.ic.cz', na Endoře 'localhost' (tohle jméno se používá poměrně často).
- Přihlašovací jméno si při aktivaci databáze buď zvolíte nebo vám bude přiděleno. Např. na IC jsem dostal 'ic_mircosoft', na Endoře se dají zakládat uživatelské účty s libovolným jménem a různými přístupovými právy.
- Heslo si při aktivaci databáze vždycky volíte jaké chcete. Doporučuji něco dlouhého, aby se to robotům špatně hádalo. Vám to vadit nebude, pamatovat si ho nemusíte - z administračního rozhraní si ho sem do zdrojáku můžete prostě zkopírovat.
- Návratová hodnota funkce je buď proměnná, přes kterou budeme ve zbytku skriptu s databází komunikovat, nebo false, když se připojení nepovede.

V praxi to může vypadat třeba takhle:

```
$spojeni=mysql_connect('localhost','moje_jmeno','StrasneTajneHe5l0');
if (!$spojeni) die('Nepodařilo se připojit k databázi.');
```

Funkce die() vypíše hlášku v závorce (podobně jako třeba echo, můžete tam dát i HTML kód) a okamžitě ukončí skript. Teoreticky bychom to dělat nemuseli, mohli bychom jenom přeskočit všechny další přístupy k databázi a vykreslit aspoň zbytek stránky, ale jestli z databáze taháme "užitečný náklad" (to, kvůli čemu sem návštěvníci přišli), nestálo by to za námahu.
Ještě jedna důležitá věc. Jak vidíte, heslo k databázi se píše přímo do zdrojáku a nedá se nijak zamaskovat. Z toho vyplývají dvě věci: za a), zdroják je za normálních okolností naprosto bezpečně skrytý, server ho ven nepouští. A za b), dávejte si velký pozor na funkce pro přímé čtení a zobrazování obsahu souborů, jako jsou show_source(), highlight_file(), file(), file_get_contents(), fread() a podobně. Jestli se nepojistíte proti PHP injekci, může se vám kdokoli dostatečně mazaný k heslu dostat.

Krok druhý: výběr databáze

Ať už máme databází jednu nebo několik, musíme systému říct, se kterou z nich chceme pracovat:

```
mysql_select_db('jmeno_databaze',$spojeni);
```

$spojeni je proměnná, kterou nám vrátila funkce mysql_connect() (tohle jméno budu používat i v celém dalším textu). Jméno databáze jsme se dozvěděli při jejím vytváření - buď jsme si ho zvolili sami nebo nám ho server přidělil.
Tyhle dva kroky (připojení a výběr) je potřeba zopakovat na začátku každého skriptu, který má s databází pracovat. Nejlepší je dát si je do jednoho malého skriptu, který pak includneme, resp. requirneme (strašné slovo), na začátek všech ostatních. Hlavní výhoda je v tom, že když potřebujeme změnit přihlašovací údaje, nemusíme je pracně přepisovat na x místech.
Tím máme úvodní administrativu za sebou, odteďka můžeme používat příkazy jazyka SQL.
Poznámka: existuje ještě příkaz mysql_close(), který spojení s databází ukončí. Praktické využití má snad jedině v případě, kdybychom uprostřed skriptu potřebovali přelézt z jednoho databázového systému do jiného, protože normálně se spojení na konci skriptu ukončuje automaticky (stejně jako se třeba mažou proměnné a jiná pomocná data). Takže se tím vůbec nemusíme zabývat.

Krok třetí: práce

Konečně se dostáváme k onomu slavnému SQL. Příkazy (dotazy) se databázi zadávají pomocí následující funkce:

```
$vysledek = mysql_query('příkaz jazyka SQL',$spojeni);
```

$spojeni je naše známá návratová hodnota z mysql_connect, tady má význam vstupního parametru a určuje, pro kterou databázi je příkaz určen.
Do proměnné $vysledek se uloží to, co nám databáze vrátí. U příkazů typu "vytvoř tabulku" nebo "vymaž řádek" tam bude jenom jednoduché true/false podle toho, jestli se to povedlo, ale u dotazů jako "dej mi všechny řádky z téhle tabulky" tam najdeme data, která se z databáze přečetla (případně false, kdyby se operace nepovedla).

Základy jazyka SQL jsou popsány v sekci MySQL databáze krok za krokem , určitě se na ně podívejte, nejsou složité Zde budeme řešit pouze to, jak s databází pracovat v PHP. Stejně si však budeme i SQL příkazy podrobně popisovat.
Příště si vytvoříme počítadlo přístupů, samozřejmě s použitím databáze.

9. díl - Počítadlo přístupů

V minulém dílu našeho PHP seriálu jsme si řekli něco o databázích. Dnes si je vyzkoušíme v praxi, vytvoříme si počítadlo návštěvníků naší stránky.

Začít musíme vytvořením tabulky. Co všechno v ní budeme potřebovat? Určitě číslo, do kterého uložíme stav počítadla, to je jasné. Počty návštěv se u začínající stránky pohybují zhruba v řádu tisíců, později desetitisíců a jestli se rozjedete opravdu hodně, můžete se dostat třeba až k milionům. Jako datový typ tedy zvolíme něco dostatečně velkého, např. INT UNSIGNED. Pro jedno počítadlo by tohle stačilo. Ale co kdybychom jich časem začali potřebovat víc, třeba kdybychom chtěli počítat přístupy na každé podstránce zvlášť? V takovém případě bude potřeba jednotlivá počítadla nějak odlišit. Můžeme jim přidělit číslo nebo třeba textové jméno, to je jedno, hlavně aby mělo každé svoje jedinečné označení. Pro ilustraci použijeme písmeno: CHAR(1). Příkaz pro vytvoření tabulky tedy bude vypadat takto:

```
CREATE TABLE pocitadla
(
id CHAR(1),
hodnota INT UNSIGNED
)
```

Tím nám vznikne tabulka jménem pocitadla, ve které budou dva sloupce: id a hodnota. Jméno id je víceméně tradice; pro identifikační klíče ho používají prakticky všichni, tak ho použijeme i my. Ale není to nutné - jestli chcete, pojmenujte si ho jinak. Praktické provedení v PHP:

```
$OK=mysql_query("CREATE TABLE pocitadla (id CHAR(1), hodnota INT UNSIGNED)",$spojeni);
if ($OK) echo 'OK, tabulka je vytvořená';
else echo 'Pozor, chyba - tabulku se nepodařilo vytvořit!';
```

Při takovéhle jednorázové akci celkem nemá cenu se piplat s kontrolou úspěšnosti, protože si tabulku můžeme ručně zkontrolovat v administračním rozhraní, které servery obvykle poskytují. Vlastně i ten příkaz pro vytvoření můžeme pustit přímo tam a ne v PHP. Je to na vás.

Dobrá, máme tabulku, ale zatím prázdnou. Co dál?

Vkládání řádků do tabulek

Příkaz se jmenuje (překvapivě) INSERT, tedy "vlož". Syntaxe vypadá takto:

```
INSERT INTO tabulka VALUES (první, druhá, třetí, ... poslední)
vlož   do       hodnoty
```

V závorce vypíšeme požadované hodnoty pro všechny položky (sloupce) vkládaného řádku, ve stejném pořadí, v jakém byly uvedeny při CREATE TABLE.

V našem případě bude vložení počítadla vypadat takhle:

```
INSERT INTO pocitadla VALUES ('A',0)
```

Počítadlo jsem si pojmenoval "A" a dal mu počáteční hodnotu 0. Zápis v PHP už si domyslete sami, návratová hodnota z mysql_query() bude opět true (povedlo se) nebo false (chyba). Tabulka tedy dopadla takhle:

id	hodnota
'A'	0

Tím je počítadlo připraveno k použití. Budeme s ním dělat celkem dvě věci: zobrazovat jeho aktuální hodnotu a zvyšovat ji o 1 při každém načtení stránky (filtrování opakovaných přístupů ze stejného počítače si necháme na jindy).

Čtení z databáze

SQL na to má příkaz SELECT ("vyber"), který z dané tabulky vybere podtabulku o zadaných rozměrech a vlastnostech. Možností, jak příkaz přesně nasměrovat na data, která chceme, je nepřeberně. Úplně nejzákladnější syntaxe vypadá takhle:

```
SELECT * FROM tabulka
vyber   z
```

Takový příkaz nám dá kompletně celý obsah tabulky. Hvězdička znamená "všechny sloupce", nepřítomnost jakýchkoli omezujících podmínek znamená, že do výběru padnou úplně všechny řádky. Na jednořádkovou tabulku s počítadlem by nám už tohle teoreticky stačilo, ale podíváme se ještě na další možnosti.

```
SELECT * FROM tabulka WHERE podmínka
vyber   z       kde
```

Tím se výběr řádků zúží pouze na ty, které splňují danou podmínku. Pro naše počítadlo by podmínka mohla být WHERE id='A' (pozor: narozdíl od PHP, znak "=" tady znamená "rovná se" a ne "přiřaď"). To už by nám určitě stačilo, ale ještě to není úplně dokonalé. Identifikační kód z tabulky číst nepotřebujeme, protože ho nehodláme zobrazovat; stačit nám bude jenom sloupec s hodnotou. Sloupce se filtrují takto:

```
SELECT sloupec1, sloupec2, ... sloupecN FROM tabulka
```

Tím tedy máme všechno, co potřebujeme. Příkaz pro načtení hodnoty počítadla A by mohl vypadat takhle:

```
SELECT hodnota FROM pocitadla WHERE id='A'
```

Teď už je přístup přes PHP samozřejmě nezbytný, protože v něm budeme zpracovávat to, co z databáze vypadne:

```
$vysledek=mysql_query("SELECT hodnota FROM pocitadla WHERE id='A'",$spojeni);
```

Důležitá je proměnná $vysledek. Do té systém uloží data, která z databáze načetl, a to ve formě tabulky (my jsme sice přečetli jenom jedno jediné číslo, ale i tak ho PHP vidí jako tabulku o velikosti 1x1). Kdyby se nic nenačetlo (třeba kdyby byla tabulka prázdná nebo kdyby žádný řádek nesplňoval zadané podmínky), vrátí se false.

Z návratových tabulek se nedá číst přímo, ale je na to celá sada speciálních funkcí:

- **mysql_fetch_assoc(tabulka)** - dá nám jeden řádek dané návratové tabulky ve formě běžného pole, indexy jsou názvy sloupců (pozor, citlivé na velikost písmen). Při prvním zavolání dá první řádek, při dalším druhý atd. a když dojede za konec tabulky, vrátí false.
- **mysql_fetch_row(tabulka)** - skoro totéž jako mysql_fetch_assoc, ale výsledné pole má číselné indexy: první sloupec má index 0, druhý 1 atd.. Je to o nepatrný ždibec rychlejší, ale o dost méně přehledné, takže bych doporučoval používat spíš fetch_assoc.
- **mysql_fetch_array(tabulka)** - kombinace předchozích dvou, výsledky můžeme indexovat jak čísly, tak jmény sloupců. V dalších příkladech budu používat prakticky výhradně tuhle funkci.
- **mysql_result(tabulka, číslo řádku, jméno sloupce)** - dá přímo hodnotu z daného řádku (počítají se od 0) a sloupce. Z hlediska rychlosti jsou výhodnější výše uvedené funkce na čtení celých řádků, ale pokud máme dlouhou

tabulku a chceme z ní třeba jenom jednu buňku, může se tohle hodit (lepší by ovšem bylo upravit Select tak, aby četl jenom to, co chceme).

- **mysql_num_rows(tabulka)** - řekne nám, kolik je v dané návratové tabulce řádků.
- **mysql_data_seek(tabulka, číslo řádku)** - posune interní "kurzor" na daný řádek (počítáno od nuly). Příští volání mysql_fetch_něco pak přečte tenhle řádek. V praxi to asi vůbec nepoužijeme.

Plus pár dalších, které nebudeme potřebovat.

Takže pokračujeme: hodnotu počítadla máme načtenou v proměnné $vysledek, teď si ji vytáhneme do obyčejného čísla:

```
$radek=mysql_fetch_array($vysledek);
if ($radek)              //Povedlo se? Kdyby ne, vyšla by logická nula.
  $cislo=$radek['hodnota'];  //OK, máme to - přečti číslo.
else $cislo='?';         //Nepovedlo se - dej tam nějakou chybovou hlášku.

echo 'Jsi náš '.$cislo.'. návštěvník!';
```

Mohli bychom použít i funkci mysql_fetch_row, v tom případě by třetí řádek vypadal takhle: $cislo=$radek[0] (jak vidíte, číslování se vztahuje na výslednou podtabulku, ne na původní tabulku, kde máme hodnotu na druhém místě).

Tím máme další krok hotový, zbývá už jenom připočítat tuhle návštěvu k uložené hodnotě.

Změny hodnot v databázi

Příkaz se jmenuje UPDATE (přeložitelné jako "změň", "uprav" nebo "aktualizuj"). Syntaxe:

```
UPDATE tabulka SET sloupec1=hodnota1, sloupec2=hodnota2 atd. WHERE podmínka
uprav          nastav
```

Nová hodnota vybraného sloupce může být jakýkoli výraz kompatibilního typu a může se v něm objevit i původní hodnota, např. cislo=10*cislo-5.

Filtrovací část s WHERE funguje stejně jako u Selectu. Když ji neuvedete, změna se provede na všech řádcích tabulky.

Přičtení jedničky k počítadlu je celkem jednoduchá věc:

```
UPDATE pocitadla SET hodnota=hodnota+1 WHERE id='A'
```

Po zabalení do funkce mysql_query příkaz vrací buď true nebo false podle toho, jestli se úprava povedla.

Tím máme počítadlo hotové, finální sesypání do jednoho skriptu už nechám na vás.

Pro úplnost doplním ještě jednu věc:

Rušení tabulek

Kdyby vás nějaká tabulka omrzela a chtěli jste ji zlikvidovat (při počátečních experimentech se to stává celkem často), dělá se to tímhle příkazem:

```
DROP  TABLE  tabulka
zahoď tabulku
```

Samozřejmě je potřeba dávat pozor, co se maže - není tady žádný čudlík "zpět", takže případný přehmat by šel napravit jenom v případě, že databázi máte někde zálohovanou.

To by pro dnešek mohlo být asi tak všechno...

Moment, a co oficiální dokumentace k MySQL?

No jo, jasně - tady je: dev.mysql.com (nezaměňujte dev za www, tím byste se dostali na komerční část stránek, odkud se k manuálům dá dopídit snad jedině přes vyhledávací okénko). V tabulce uprostřed stránky si vyberte svůj oblíbený jazyk a datový formát, klikněte a jste tam.

A to už je opravdu všechno.

Komu to nestačilo, může se podívat na alternativní verzi tohoto trojčlánku s několika drobnostmi navíc.

Příště se do SQL ponoříme hlouběji a vytvoříme si webovou anketu.

10. díl - Anketa - Tvoříme anketu

Vítejte u dalšího pokračování seriálu o tvorbě dynamických doplňků na webové stránky. Dnes se naučíme pracovat s IP adresami, probereme základy zabezpečení SQL a pár nových triků a funkcí.

Připomínám, že jednotlivé díly seriálu na sebe navazují a k jednou probrané látce už se nevracím. Začátečníkům proto doporučuji číst od začátku, jinak nemůžu zaručit, že se v tom vyznají.

Začněme rovnou příkladem

Minule jsme si napsali jednoduchoučké počítadlo přístupů, které při každém načtení stránky zvětšilo jedno číslo v databázi o 1 a zobrazilo ho. Tentokrát se pustíme do něčeho složitějšího: do ankety. V našem případě to bude udělátko, které vypíše jednu otázku a několik možných odpovědí a když některou z nich návštěvník vybere, v příslušné škatulce přibude jeden bod a anketa se zamkne, aby jeden člověk nemohl hlasovat víckrát než jednou.

Co k tomu budeme potřebovat?

Řekněme, že anket budeme chtít mít na stránce několik, vzájemně nezávislých a s různými otázkami i odpověďmi. To znamená, že musíme navrhnout tabulky, do kterých se nám vejde toto:

- Text otázky, pro každou anketu jeden.
- Texty odpovědí, pro každou anketu obecně libovolný počet.
- Počty hlasů u jednotlivých odpovědí, tj. čísla, kterých bude stejný počet jako textů.
- Pojistka proti vícenásobnému hlasování, pro každou anketu zvlášť.

Text otázky bude nějaký vhodný textový řetězec (stačí velice krátký, třeba TINYTEXT nebo nějaký VARCHAR), plus nějaký identifikační kód, který určuje, ke které anketě otázka patří. Dejme tomu, že si ankety budeme číslovat, takže ten kód může být třeba celé číslo; předpokládám, že nebudeme provozovat tisíce anket najednou, takže nám postačí i ten nejmenší TINYINT. Pro účely optimalizace ho můžeme označit za primární klíč (PRIMARY KEY), pro účely pohodlí můžeme ještě dodat AUTO_INCREMENT, aby se každé nově přidané anketní otázce automaticky přiřadil kód o 1 větší než té předchozí, ale nutné to není - pokud anket není moc, může být výhodnější číslovat si je ručně.

Odpovědi do stejné tabulky nacpat nemůžeme, protože nevíme, kolik jich ve které anketě bude, a proměnný počet sloupců se udělat nedá. Sice bychom si mohli připravit sloupce třeba pro dvacet odpovědí a u menších anket některé z nich nechat nevyužité, ale podle zákona schválnosti bychom stejně jednou zjistili, že jich potřebujeme nejméně dvacet jedna. Takže to uděláme jinak, odpovědi půjdou do samostatné tabulky.

Texty odpovědí a počty hlasů bude nejlepší držet hezky pohromadě v jedné tabulce. Odpověď bude zase nějaký krátký text, počet hlasů nějaké dostatečně velké přirozené číslo - třeba INT UNSIGNED. Potom samozřejmě potřebujeme kód ankety, abychom věděli, ke které to patří.

Mimochodem, tím nám vznikne přímo ukázkový vztah neboli relace (odtud pojem "relační databáze") mezi tabulkou otázek a tabulkou odpovědí. Vztah je typu 1:n, tedy k jedné otázce z první tabulky se váže libovolný počet odpovědí z tabulky druhé. Zároveň si tohle uspořádání můžeme představit i jako stromovitou hierarchickou strukturu: otázku jako kořen a odpovědi jako

větve na stejné úrovni. To je dobré si zapamatovat: jakoukoli hierarchii můžeme v relační databázi nasimulovat tak, že si každý uzel pamatuje svého jediného nadřízeného.

Konec terminologické odbočky, zpátky k tabulce. Bude nám text, počet hlasů a kód ankety stačit? Teoreticky možná ano, ale je potřeba si uvědomit, že až návštěvník odešle svůj hlas, budeme ho muset nějak jednoznačně a pokud možno úsporně identifikovat. Posílat přes odkaz nebo formulář celý text odpovědi je krajně nepraktické, nehledě na to, že texty klidně můžou být v několika anketách stejné (jako třeba odpověď "Ano"). Také potřebujeme něco, podle čeho by se odpovědi v anketě řadily - ne vždy to chceme podle abecedy. Takže přidáme ještě unikátní identifikační kód, nejlépe číselný (v takovém případě můžeme s výhodou použít auto_increment, protože na odpovědi ruční číslování určitě potřebovat nebudeme).

Pojistku proti vícenásobnému hlasování si necháme na později, první pokus si pro jednoduchost napíšeme bez ní. Předem prozradím, že jenom přidáme třetí tabulku a pár podmínek navíc, první dvě tabulky zůstanou beze změny.

Takhle nějak by tedy mohly vypadat příkazy pro vytvoření tabulek otázek a odpovědí:

```
CREATE TABLE otazky
(
Otazka TINYTEXT,
KodAnkety TINYINT PRIMARY KEY
)

CREATE TABLE odpovedi
(
id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
KodAnkety TINYINT,
PocetHlasu INT UNSIGNED,
Odpoved TINYTEXT
)
```

Místo Tinytextu jsme stejně dobře mohli použít VARCHAR(něco), klidně i kratší než těch 255 znaků. Také jsme mohli u všech důležitých položek (kromě klíčů, tam je to zbytečné) dodat NOT NULL, aby se nedaly vložit prázdné. Ale to už jsou jenom drobnosti, které na další postup nemají žádný vliv. Na velikosti písmen v názvech sloupců nezáleží, takže např. PocetHlasu můžeme později psát třeba samými malými písmeny, nebude to vadit.

Tabulky máme, co dál?

Příprava ankety k použití

Dejme tomu, že na svých stránkách o lehké atletice chceme rozjet anketu o nejoblíbenější sport. Otázka je tedy celkem jasná, kód ankety si zvolíme třeba 1, protože je to naše první anketa. Vložení otázky do tabulky, by mohlo vypadat např. takto:

```
INSERT INTO otazky VALUES ('Jaký je váš nejoblíbenější sport?',1)
```

Vkládání odpovědí do druhé tabulky není o moc složitější:

```
INSERT INTO odpovedi VALUES (NULL, 1, 0, 'Lyžování');
INSERT INTO odpovedi VALUES (NULL, 1, 0, 'Plavání');
INSERT INTO odpovedi VALUES (NULL, 1, 0, 'Běhání');
INSERT INTO odpovedi VALUES (NULL, 1, 0, 'Geohashing');
INSERT INTO odpovedi VALUES (NULL, 1, 0, 'Gaučing');
```

Hodnota NULL v prvním sloupci bude díky auto_incrementu automaticky nahrazena vzestupnou číselnou řadou (1, 2, 3...). Jednička u kódu ankety je jasná - odpovědi patří k výše uvedené otázce číslo jedna. Nula je počáteční počet hlasů, teoreticky bychom sem mohli dát i nějaké jiné číslo (pokud třeba přenášíme hlasy odjinud nebo pokud chceme statistiku trochu popostrčit vhodným směrem ). Text odpovědi je jasný.

A teď co s tím. Buď tyto příkazy pustíme přímo přes nějaké webové rozhraní k MySQL (např. PHPMyAdmin), které obvykle bývá k dispozici; v takovém případě jenom pozor na kódování české diakritiky. Nebo bychom mohli každý řádek obalit do mysql_query, všechno uložit do jednoho PHP skriptu, ten nahrát na server a přes prohlížeč spustit. Tím by sice odpadly potenciální problémy s češtinou, ale kdo se s tím má pořád tak zdlouhavě patlat. Protože lenost je matka pokroku a protože se tím naučíme spoustu nových věcí, vyrobíme si na tvorbu anket administrátorské rozhraní se vším všudy. Ale protože to není ani nezbytně nutné ani úplně triviální, necháme si to až úplně na konec a nejdřív si napíšeme skript, který anketu zobrazí a zpracuje hlasy od návštěvníků.

Zobrazení ankety

Pro jednoduchost zatím necháme stranou estetickou stránku věci (obvyklý rámeček kolem ankety a podobně) a vypíšeme jenom nejdůležitější údaje uspořádané v neviditelné tabulce. První řádek (resp. záhlaví) tabulky bude vyhrazen pro otázku. Další řádky budou rozdělené do dvou buněk, v levé bude text odpovědi zároveň fungující jako klikatelný odkaz pro odeslání hlasu a v pravé bude aktuální počet hlasů pro tuto odpověď. Dopadnout to může dejme tomu takhle:

```
$cislo=1; //číslo ankety
$vysledek=mysql_query("SELECT otazka FROM otazky WHERE kodankety=$cislo",$spojeni);
$radek=mysql_fetch_row($vysledek);
echo '<table><tr><td colspan="2">'.$radek[0].'</td></tr>';
```

Proměnnou $cislo jsme si zavedli proto, aby se následující kód nemusel opisovat pro každou anketu zvlášť. Nakonec si z toho stejně uděláme funkci a číslo jí můžeme pohodlně předat jako parametr.

Druhým příkazem jsme si do proměnné $vysledek z databáze vytáhli všechny otázky, které mají kód ankety 1. Taková je samozřejmě jenom jedna, takže výsledek bude tabulka o velikosti 1x1. $spojeni je jako obvykle proměnná, kterou jsme dostali od funkce mysql_connect.

Třetí příkaz z návratové tabulky vytáhne první (a zároveň i jediný) řádek. Bude to pole všehovšudy s jedním prvkem, takže nám nehrozí popletení indexů a je jednodušší použít fetch_row, která dává pole indexované čísly.

Poslední příkaz vypíše HTML kód pro první řádek tabulky a do něj vloží z připraveného pole první (resp. nultou) položku - text otázky.

Pozn.: Uvnitř řetězců v uvozovkách se jména proměnných automaticky nahrazují jejich hodnotou, takže místo textu "$cislo" se v příkazu objeví jednička.

Otázka byla jednoduchá, teď odpovědi:

```
$vysledek=mysql_query("SELECT id, pocethlasu, odpoved FROM odpovedi WHERE kodankety=$cislo ORDER BY id ASC",
$spojeni);
while ($radek=mysql_fetch_array($vysledek))
{
```

```
echo '<tr><td>';
echo '<a href="anketa.php?hlasujpro='.$radek['id'].'">'.$radek['odpoved'].'</a>';
echo '</td><td>';
echo $radek['pocethlasu'];
echo '</td></tr>';
};
echo '</table>';
```

Pozor, tady se nám v SQL objevila nová konstrukce: ORDER BY neboli "seřaď podle". Tím jsme řekli, že v návratové tabulce $vysledek chceme mít řádky uspořádané podle hodnoty ve sloupci id, a to vzestupně (ASC jako ascending). Vzestupně znamená, že první řádek bude mít id nejmenší a poslední největší. Opačně by bylo DESC (descending). ASC je výchozí hodnota, takže by nám stačilo napsat jenom order by něco. Řadit samozřejmě můžeme podle čehokoli (texty by se braly abecedně). Kdybychom řazení neuvedli, výběr by sice asi přišel v takovém pořadí, v jakém byl do databáze vložen (což je to, co teď zrovna chceme), ale obecně se na to nedá moc spoléhat.

Následuje cyklus, který bude postupně vybírat z tabulky výsledků jednotlivé řádky a předávat je přes pole $radek, odkud si je vyzvedneme a po jednotlivých položkách zobrazíme.

Tvar hlasovacího odkazu jsem si zvolil, teoreticky by nemusel vypadat zrovna takhle. Po kliknutí přejde na skript anketa.php, což bude předpokládám přímo tenhle, který zrovna píšeme. Zároveň metodou Get předá proměnnou $_GET['hlasujpro'] a v ní hodnotu položky id od dané odpovědi.

Nakonec uzavřeme tabulku a jsme hotovi.

Zpracování hlasu

Návštěvník naší stránky si tedy nechal zobrazit anketu, klikl na jednu odpověď a my se s ní teď musíme vypořádat. Uděláme vlastně úplně totéž, jako v případě počítadla - najdeme v databázi jedno číslo a zvýšíme ho o 1:

```
if (isset($_GET['hlasujpro']))
mysql_query("UPDATE odpovedi SET pocethlasu=pocethlasu+1 WHERE id=".$_GET['hlasujpro'],$spojeni);
```

Ovšem **POZOR**, tentokrát je v tom jeden velikánský háček: co kdyby nějakého chytráka napadlo prohlédnout si text odkazu, zamyslet se a do adresního řádku ručně naťukat třeba tohle:

```
http://nase.stranka.cz/anketa.php?hlasujpro=1%20OR%205%3d5
```

%20 je kód mezery a %3d kód rovnítka, rozkódování proběhne automaticky. Po přímém dosazení do příkazu by databáze dostala tohle:

```
UPDATE odpovedi SET pocethlasu=pocethlasu+1 WHERE id=1 OR 5=5
```

Podtržením je zvýrazněn celý zadaný vstup. Nepříjemné, že? Odkliknutí všech odpovědí ve všech anketách sice nemá žádný praktický smysl, ale zkuste si představit, že by takhle někdo převezl třeba podmínku testující administrátorské heslo.

Této technice útoku se říká **SQL injection** a je to asi nevětší nebezpečí, s jakým se v běžném životě setkáme a kterému se musíme vyhnout.

V našem případě je řešení celkem jednoduché. Očekáváme číselný kód, ale text '1 OR 5=5' má k číslu daleko, tak ho můžeme rovnou vyloučit. Kontrolu číselnosti zajistí funkce **is_numeric**, která vrací true v případě, že se parametr skládá pouze z číslic, případně s nějakým tím znaménkem nebo desetinnou tečkou:

```
if (is_numeric($_GET['hlasujpro']))
mysql_query("UPDATE... atd., viz výše);
else die('Co to na mě zkoušíš, podvodníku?');
```

Jiným způsobem zabezpečení je tzv. escapování, kdy se před všechny mezery, středníky, apostrofy, uvozovky a podobné potenciálně nebezpečné znaky předřadí znak \ (zpětné lomítko). Ale to má smysl především pro texty a podrobně si ho probereme někdy příště.

11. díl - Anketa - Vylepšujeme anketu

Pojistka proti vícenásobnému hlasování

Anketa by nám teď měla fungovat, ale má jednu nevýhodu: kdokoli do ní může naklikat libovolný počet jakýchkoli hlasů. To se nám ale nelíbí, chtěli bychom, aby každý mohl hlasovat jenom jednou. Jak to udělat?

Máme v podstatě dvě možnosti: buď si zapamatovat IP adresu hlasujícího počítače, nebo v něm nechat sušenku (cookie). Cookies mají výhodu v jednoznačnosti (nastaví se opravdu jenom na ten jeden počítač, ze kterého se hlasovalo, a nezablokuje anketu jiným), ale nevýhodu v tom, že si je uživatel může kdykoli smazat a hlasovat znovu. IP adresy sice nemusí být jednoznačné (víc počítačů může mít stejnou) ani trvalé (na stejném počítači se může měnit), ale zase jsou jednodušší na obsluhu a tak nějak blbuvzdornější. Proto si první ochranu založíme právě na IP adresách.

Postup bude celkem jednoduchý: vytvoříme si další tabulku, do které budeme ukládat IP adresy počítačů, ze kterých se hlasovalo, spolu s kódem ankety, aby nám odhlasování v jedné nezablokovalo všechny ostatní:

```
CREATE TABLE ipadresy
(
ip VARCHAR(15),
kod TINYINT
)
```

Položka ip je vlastní IP adresa. Od serveru ji dostaneme dostaneme v podobě textového řetězce obsahujícího čtyři čísla v rozsahu 0..255, oddělená tečkami (např. '123.255.20.1'). Varchar(15) jsem si zvolil proto, že na víc než 15 znaků to vyjít nemůže; teoreticky by stejně dobře posloužil třeba Tinytext.

Položka kod odpovídá kódu ankety. Záměrně jsem ji nepojmenoval KodAnkety, protože tuhle tabulku můžeme zároveň využít třeba pro počítadla přístupů (viz minule), stačí jim přidělit nějaké kódy, které se nebudou tlouct s existujícími anketami. Ale to už nechám na vás.

Aktuální IP adresu návštěvníka najdeme v superglobálním poli $_SERVER, konkrétně pod názvem $_SERVER['REMOTE_ADDR'] (pozor, všechna písmena jsou velká!). To je všechno, co potřebujeme, můžeme se pustit do práce. Po odhlasování uložíme adresu do tabulky následujícím způsobem:

```
mysql_query("INSERT INTO ipadresy VALUES ('".$_SERVER['REMOTE_ADDR']."',$cislo)",$spojeni);
```

Povšimněte si apostrofů připravených okolo místa, do kterého vkládáme adresu - bez nich by to nešlo, je to text. Vzhledem ke složitosti zápisu téhle proměnné jsem ji radši připojil pomocí teček, automatickému rozbalování zas až tak moc nevěřím :-]. $cislo na konci je kód ankety. Ovšem pozor - kde jsme ho vlastně vzali? V tuhle chvíli máme po ruce jenom id odeslané odpovědi, proměnnou $_GET['hlasujpro']. Nezbývá, než si ho ještě před pokusem o uložení ajpiny v příslušné tabulce najít:

```
$vysledek=mysql_query("SELECT kodankety FROM odpovedi WHERE id=".$_GET['hlasujpro'],$spojeni);
if ($radek=mysql_fetch_row($vysledek)) $cislo=$radek[0];
else die('Tuhle odpověď nemáme v databázi!');
```

Prvním příkazem jsme získali buď tabulku 1×1 obsahující hledaný kód, nebo nic, pokud odpověď s daným id nebyla v databázi nalezena (to se může stát snad jenom při ručním hraní s adresním řádkem). Samozřejmě předpokládám, že proměnnou $_GET['hlasujpro'] už touhle dobou máme **důkladně zkontrolovanou** a víme, že je to platné číslo. Jestli ne, ať vás ani nenapadne strkat ji do SQL!

Druhý příkaz se pokusí z načtené jednobuňkové minitabulky vytáhnout hledaný kód. Kdyby se mu to nepovedlo, zahlásí chybu a ukončí skript (to samozřejmě není jediná možnost, jak se s chybou vyrovnat).

Shrneme to: podle čísla odpovědi jsme si našli kód ankety a spolu s IP adresou jsme ho uložili do databáze. Tím máme hotovou poslední fázi zpracování hlasu. Ještě se ale musíme vrátit na začátek, do okamžiku, kdy jsme dostali číslo odpovědi, zkontrolovali ho a teď se rozmýšlíme, jestli tenhle hlas započítáme nebo jestli už má daná adresa odhlasováno. Bude to chtít kontrolu, jestli už máme tuhle adresu uloženou a hlas zahodíme, nebo jestli ji ještě nemáme, takže ji uložíme a hlas započítáme:

```
$vysledek=mysql_query("SELECT * FROM ipadresy WHERE ip='".$_SERVER['REMOTE_ADDR']."' AND kod=".$cislo,
$spojeni);
if (mysql_num_rows($vysledek)==0) ...tenhle tu ještě nebyl, jeho hlas uložíme...
else ...už ho tu máme uloženého, další hlasy od něj ignorujeme...
```

První příkaz zkusí z tabulky IP adres vytáhnout kombinaci zpracovávané ankety a aktuální adresy návštěvníka. Pokud tam taková kombinace není (tj. ještě nebylo hlasováno), vrátí prázdnou tabulku s nulovým počtem řádků. Pokud tam je, vrátí nám jednořádkovou tabulku. Dovnitř do ní vůbec nemusíme koukat (stejně víme, co tam je - přesně to, co jsme zadali do té podmínky). Pozor, nestačí ověřit $vysledek na true nebo false - true dostaneme vždycky, i když se nevrátí žádná data! False by se objevilo jenom při chybě.

Pozn.: teoreticky samozřejmě můžeme místo té hvězdičky napsat jméno některého sloupce, ale výsledný efekt by byl stejný: buď dostaneme něco nebo nic.

Poslední drobnost, ke které se uložené adresy dají použít, je to, že anketu, ve které už se nedá hlasovat, vykreslíme rovnou bez klikatelných odkazů, aby to návštěvníky nemátlo - takovéhle věci je lepší se dozvědět na první pohled a ne až po zdlouhavém kliknutí. To bude záležitost toho kousku, kde vypisujeme jednotlivé odpovědi: prostě v takovém případě vynecháme <a>....

Stop, mám v tom guláš! Jak to všecho patří dohromady?

Pravda, zatím jsme probírali jednotlivé detaily a celek se ztrácí kdesi v nedohlednu. Tak tedy: všechno máme v jednom skriptu, který dělá tohle:

1. Přijetí hlasu ($_GET['hlasujpro']). Pokud přišel, zkontrolujeme jeho platnost (is_numeric), najdeme k němu kód ankety a obě čísla si uložíme do nějakých vhodných proměnných. Pokud nepřišel nebo není platný, přeskočíme jeho zpracování.
2. Kontrola, jestli tenhle návštěvník ve zvolené anketě může hlasovat.
3. Zpracování hlasu (pokud přišel, je platný a může se hlasovat):
    1. Zvyš počet hlasů u zvolené odpovědi.
    2. Návštěvníkovu IP adresu ulož do tabulky adres.
4. Cyklus pro každou anketu, kterou na téhle stránce máme:
    1. Zkontroluj, jestli tenhle návštěvník v téhle anketě může hlasovat.
    2. Pokud může, vypiš anketu s odpověďmi ve formě aktivních odkazů. Pokud ne, vypiš odpovědi jako neaktivní text.

Jak vidíte, kontrolovat přítomnost adresy v tabulce musíme nejméně dvakrát: jednou na začátku skriptu při zpracovávání hlasu a potom jednou pro každou vykreslovanou anketu.

Kosmetické detaily

Počty hlasů v anketách se obvykle znázorňují graficky pomocí různých sloupečků nebo barevných žížalek. To by v tom byl čert, abychom to nedokázali taky!

Teoreticky by se vodorovný pruh asi dal vyrobit i pomocí vhodně ostylovaných HTML elementů hr nebo div, ale nejmodernější CSS není moje silná stránka a navíc by to nebylo zrovna nejpružnější, takže použijeme klasické obrázky - element img. Využijeme toho, že se obrázku dají nastavit libovolné rozměry a nemusí se dodržovat ani poměr stran. To znamená, že můžeme mít fyzicky uložený jenom krátký úsek (jestli nepotřebujeme barevné přechody mezi levým a pravým koncem, může to být třeba jenom jednopixelová nudle) a roztáhneme si ho podle potřeby. Pokud máme konce proužku nějak tvarově odlišené (kulaté, stínované, plastické apod.), roztažení by jim uškodilo, takže je musíme uložit zvlášť. Měli bychom tedy celkem tři malé obrázky: pevný levý konec, roztažitelný střed a pevný pravý konec. Dejme tomu, že je máme na serveru uložené pod jmény levykonec.gif, stred.gif a pravykonec.gif a že jsou všechny 10 pixelů vysoké.

První otázka je, jak dlouhé proužky udělat. Nabízí se triviální možnost 1 hlas = 1 pixel, ale to by mělo dvě nevýhody: zaprvé by při malých počtech hlasů byly proužky moc krátké a rozdíly v délkách by nebyly pouhým okem viditelné, zadruhé by se naopak při hodně vysokých počtech anketa neomezeně roztahovala. Potřebujeme tedy nějaký přepočet, který maximální délku omezí a zároveň zajistí, aby každý hlas byl vidět. Nejjednodušší je prohlásit, že odpověď s největším počtem hlasů bude odpovídat maximální šířce proužku, a všechny ostatní šířky trojčlenkou smrsknout nebo roztáhnout v příslušném poměru (tak to dělají například diskusní fóra PHPBB):

$$\frac{\text{max. počet hlasů}}{\text{max. šířka proužku}} = \frac{\text{nějaký jiný počet hlasů}}{\text{hledaná šířka proužku}}$$

neboli po úpravě:

hledaná šířka proužku = nějaký jiný počet hlasů * max. šířka proužku / max. počet hlasů

Jediná výjimečná situace, na kterou si musíme dát pozor, je počáteční stav ankety, kdy jsou všechny počty a tedy i maximum nulové. Asi nemusím připomínat, že dělení nulou by nedopadlo dobře 😉.

Teď ještě jak to naprogramovat (v proměnné $cislo máme kód ankety):

```
$vysledek=mysql_query("SELECT MAX(pocethlasu) FROM odpovedi WHERE kodankety=$cislo",$spojeni);
if (!($radek=mysql_fetch_row($vysledek)) or (($maximum=$radek[0])==null))
die('Tuhle anketu nemáme na skladě.');
```

Tady se nám objevuje nová funkce: MAX(sloupec) nám dá největší hodnotu z daného sloupce, výsledek dostaneme ve formě jednobuňkové návratové tabulky. Pokud by dané podmínce (where) neodpovídaly žádné řádky a maximum tedy nebylo z čeho počítat, v návratové tabulce bude hodnota null (pozor, návratová tabulka by existovala a obsahovala by hodnotu, akorát že by ta hodnota byla null - nestačí otestovat fetch na true/false). Obdobně funguje MIN(), která dává minimum.

Další věc, která možná potřebuje trochu vysvětlit (aspoň pro ne-céčkaře), je ten divoký logický výraz v závorce ifu. Takže: nejdřív se do proměnné $radek načte obsah návratové tabulky. Kdyby byla prázdná, celý ten přiřazovací výraz by dal hodnotu false a po negaci true. Druhá část podmínky by se vůbec neuplatnila (na výsledku oru by stejně nic nezměnila), takže by se rovnou provedlo die(). Jestli prázdná nebyla, pokračuje se druhou podmínkou. V ní se naplní proměnná $maximum, výsledná

hodnota tohoto přiřazovacího výrazu (která je rovná tomu, co bylo přiřazeno) se hned porovná s nullem a jestli souhlasí, vyjde logické true a voláme die.
Na hlavičku ankety se naše kosmetické úpravy nevztahují, tak se mrkneme rovnou na odpovědi. Dejme tomu, že proužek zobrazíme ve stejné buňce přímo nad otázkou. Zároveň také rovnou provedeme dříve zmíněné vynechání odkazů, pokud už se hlasovalo (to je ta proměnná $muze_hlasovat; předpokládám, že už ji máme připravenou z dřívějška):

```
$vysledek=mysql_query("SELECT id, pocethlasu, odpoved FROM odpovedi WHERE kodankety=$cislo ORDER BY id ASC",
$spojeni);
while ($radek=mysql_fetch_array($vysledek))
{
echo '<tr><td>';
//proužek:
echo '<img src="levykonec.gif">';
if ($maximum==0) $sirka=0; //pojistka proti dělení nulou
else $sirka=round($radek['pocethlasu']*200/$maximum);
echo '<img src="stred.gif" height="10" width="'.$sirka.'">';
echo '<img src="pravykonec.gif"><br>';
//otázka a počet hlasů:
if ($muze_hlasovat) echo '<a href="anketa.php?hlasujpro='.$radek['id'].'">';
echo $radek['odpoved'];
if ($muze_hlasovat) echo '</a>';
echo '</td><td>';
echo $radek['pocethlasu'];
echo '</td></tr>';
};
```

Zakončení tabulky a podobné formality už nechám na vás.
200 je požadovaná maximální šířka proužku v pixelech. Hodnotu jsem si zvolil, může to být samozřejmě i cokoli jiného. 10 je požadovaná výška proužku; zadat se musí, jinak by většina prohlížečů roztáhla obrázek i svisle, aby se zachoval poměr stran. Výška proužku by pochopitelně měla být stejná jako výška levého a pravého konce, na které má navazovat.
Použili jsme novou funkci: **round**(číslo), která dané číslo zaokrouhlí na celé. To je nutné, protože šířka obrázku musí být zadána v celých pixelech.

Poslední věc, která stojí za zmínku, je možnost zabalit obsluhu anket do funkcí (viz dříve). Pokud si napíšeme jednu funkci pro zpracování hlasu a druhou pro zobrazení jedné ankety (její kód by dostala v parametru), výrazně si zpřehledníme zápis v hlavní části skriptu. Navíc pokud se nám podaří po vyhodnocení hlasu skočit zpátky na původní stránku (napadá mě předat jméno cílové stránky jako další parametr zobrazovací funkce, ale možná existuje i něco pohodlnějšího), můžeme si anketní funkce schovat do samostatného souboru a ten pak podle potřeby includovat a využívat na všech ostatních stránkách.
Tím jsme s provozní částí anket víceméně hotovi. Zbývá sestavit si nějaký skript, který by nám usnadnil jejich tvorbu a úpravy. Pokračujte prosím tudy.

12. díl - Anketa - Administrační rozhraní

Anketní otázky a odpovědi sice můžeme tvořit a upravovat přímo pomocí příkazů SQL, ale to je poměrně pracné a hrozí, že se někde překlepneme. Pokud jste dost líní na to, abyste investovali nemalé množství práce do stránky, která vám následně trošku práce ušetří, čtěte dál .
Co všechno budeme potřebovat?

- Formulář na vytvoření nové ankety. Bude v něm textové políčko pro napsání otázky a políčko pro napsání kódu (tedy pokud si ho nenecháváme generovat auto_incrementem - předpokládám, že ne).
- Formulář pro výběr ankety k úpravám. To může být roletová nabídka, sada odkazů nebo cokoli, co vás napadne. Pro jednoduchost to předvedu na těch odkazech.
- Formulář pro přidání nové odpovědi. Jedno prázdné textové políčko. Protože ho budeme používat ze všech nejčastěji, mělo by být na stránce někde nahoře.
- Formulář pro editaci otázky a odpovědí vybrané ankety. V něm bude jedno textové políčko pro každou odpověď, původní texty v nich budou předvyplněné. U každé odpovědi bude odkaz na její smazání, u otázky pak jeden na smazání celé ankety.

Každý formulář samozřejmě bude mít i svoje odesílací tlačítko.
V následujícím textu postupně uvedu jednotlivé formuláře a příslušné vyhodnocovací algoritmy. Předpokládám, že všechno pojede v jednom skriptu nazvaném tvorbaanket.php. O rozložení na stránku, vzhled a podobné nepodstatné drobnosti už se touhle dobou doufám dokážete postarat sami .
Zatím nebudeme řešit přihlašování, ochranu heslem, antispam a podobné věci. Předpokládám, že tenhle jednoúčelový skript budeme nahrávat na server vždycky jenom na chvíli a po použití ho zase smažeme, takže se k němu nikdo nestihne dostat a zneužít nezabezpečený přístup. Ze stejného důvodu se protentokrát můžeme vykašlat i na ochranu proti SQL injekci.

Vytvoření nové ankety

Formulář:

```
<form action="tvorbaanket.php" method="post">
<input type="text" name="tvorotazka" value="">
<input type="text" name="tvorkod" value="">
<input type="submit" value="Vytvoř anketu">
</form>
```

Vyhodnocení:

```
if (isset($_POST['tvorotazka']) and isset($_POST['tvorkod']))
{
//kontrola, jestli uz nahodou neexistuje:
$vysledek=mysql_query("select * from otazky where kodankety=".$_POST['tvorkod'],$spojeni);
if (mysql_num_rows($vysledek)==0)
{
if (mysql_query("insert into otazky values (".$_POST['tvorkod'].",'".$_POST['tvorotazka']."')",$spojeni))
echo 'Anketa vytvořena.';
```

```
else echo 'Anketu se nepodařilo vytvořit.';
};
else echo 'Anketa s tímhle kódem už existuje, zkus zadat nějaký jiný.';
};
```

Pozn.: z hlediska pohodlí by bylo nejlepší, aby se nově vytvořená anketa hned automaticky vybrala pro úpravy. Ale to už nechám na vás. Nápověda: vyhodnotíme výstup a pak pomocí příkazu header (viz dříve) skočíme na stránku s adresou upravenou podle následujícího odstavce.

Seznam anket s výběrem

Dejme tomu, že to bude jednoduchý seznam odkazů. Teď ale pozor: informaci o tom, která anketa je vybraná, musíme někudy předat. S odkazy nemáme jinou možnost než metodu Get neboli vepsání hodnoty přímo do adresy stránky. Dejme tomu, že si tu proměnnou pojmenujeme vybranaanketa. Hlavně je potřeba nezapomenout, že ji pak musíme neustále předávat i z dalších formulářů, jinak by se nám výběr ztratil.

Nejdřív vyhodnocení - tohle musíme připravit vždycky:

```
$vybranaanketa = isset($_GET['vybranaanketa']) ? $_GET['vybranaanketa'] : '';
```

Pro ilustraci jsem použil céčkovský špek zvaný **podmíněný výraz**. Obecně se zapisuje jako Podmínka?Hodnota1:Hodnota2 a znamená: pokud je daná Podmínka pravdivá (true), výsledkem výrazu je Hodnota1, jinak Hodnota2. Co se týče priority těchto operátorů, je úplně nejnižší. To mimo jiné znamená, že jestli chcete třeba sestavit řetězec pomocí operátoru "." a jako některé části použijete podmíněné výrazy, musíte je uzavřít do (závorek), jinak by se tečky vyhodnotily jako první, přilepily by vám kousky řetězce k podmínce a druhé hodnotě a vyšel by vám z toho nesmysl.

A teď ten seznam:

```
$vysledek=mysql_query("select * from otazky",$spojeni);
if (mysql_num_rows($vysledek)==0)
{
echo '<p>Zatím tu žádnou anketu nemáme.</p>';
else:
echo '<ul>';
while ($radek=mysql_fetch_array($vysledek))
if ($vybranaanketa==$radek['kodankety'])
echo '<li>'.$radek['kodankety'].': '.$radek['otazka'].' (aktuálně vybraná)</li>';
else echo '<li><a href=tvorbaanket.php?vybranaanketa='.$radek['kodankety'].'">'
.$radek['kodankety'].': '
.$radek['otazka'].'</a></li>';
echo '</ul>';
};
```

Přidání otázky k vybrané anketě

Celkem jednoduchá záležitost. Pozor je potřeba dát jenom na správné ošetření vybrané ankety (předpokládám, že proměnnou $vybranaanketa máme připravenou z předchozího odstavce). Formulář:

```
if ($vybranaanketa<>'')
{
echo '<form action="tvorbaanket.php?vybranaanketa='.$vybranaanketa.' method="post">';
echo '<input type="text" name="pridejodpoved" value="">';
echo '<input type="submit" value="Přidej odpověď">';
echo '</form>';
else:
echo 'Žádná anketa není vybraná.';
};
```

Tady celkem není co řešit, snad jenon by šlo doplnit políčko pro zadání počátečního počtu hlasů. Ale to už je detail.

Vyhodnocení:

```
if (($vybranaanketa<>'') and isset($_POST['pridejodpoved']) and ($_POST['pridejodpoved']<>''))
if (mysql_query("insert into odpovedi values (null, $vybranaanketa, 0, '".$_POST['pridejodpoved']."')",$spojeni))
echo 'Odpověď vložena.';
else echo 'Odpověď se nepodařilo vložit.';
```

Hlášení o úspěšnosti je víceméně zbytečné (odpověď buď uvidíme v následujícím formuláři nebo ne), dal jsem ho sem jenom pro úplnost.

Úpravy vybrané ankety

Tohle je nejsložitější část celého skriptu, ale v podstatě není úplně nezbytně nutná, takže jestli chcete, můžete ji vynechat (jenom to mazání byste pak museli dělat ručně).

Formulář:

```
if (($vybranaanketa<>'')
{
echo '<form action="tvorbaanket.php?vybranaanketa='.$vybranaanketa.'" method="post">';
//otázka:
$vysledek=mysql_query("select otazka from otazky where kodankety=$vybranaanketa",$spojeni);
if ($radek=mysql_fetch_row($vysledek))
echo 'Otázka: <input type="text" name="editotazka" value="'.$radek[0].'"> '
.'<a href="tvorbaanket.php?vybranaanketa='.$vybranaanketa.'&smazanketu=jo">Smaž celou tuhle anketu</a>';
else echo 'Vybraná anketa neexistuje.';
//odpovědi:
echo '<br>Odpovědi:<br>';
$vysledek=mysql_query("select id, odpoved from odpovedi where kodankety=$vybranaanketa order by id asc",$spojeni);
while ($radek=mysql_fetch_array($vysledek))
{
echo '<input type="text" name="editodpoved'.$radek['id'].'" value="'.$radek['odpoved'].'"> '
.'<a href="tvorbaanket.php?vybranaanketa='.$vybranaanketa.'&smazodpoved='.$radek['id'].'">Smaž tuhle
odpověď</a><br>'
};
```

```
echo '<input type="submit" value="Ulož změny">';
echo '</form>';
};
else echo 'Žádná anketa není vybraná.';
```

Jak vidíte, metody Get a Post se dají kombinovat: kód vybrané ankety posíláme přes adresu (Get) a obsah textových políček jinudy (Post).

Otázku si nejdřív vytáhneme z databáze. Pokud ji tam najdeme (jako že asi jo, jinak bychom se na ni nemohli přes seznam doklikat), vypíšeme ji v textovém políčku. K ní ještě přidáme odkaz na smazání ankety - ten předá proměnnou $_GET['smazanketu'] s hodnotou 'jo'. Na hodnotě nám nezáleží, anketu ke smazání už nám jednoznačně určuje proměnná $vybranaanketa.

Odpovědi se od otázky zas až tak moc neliší, jenom je jich víc, takže musíme použít cyklus. Také je potřeba jednotlivá políčka navzájem nějak odlišit: na to použijeme položku id, kterou připojíme ke jménu (name). Mazací odkaz funguje podobně jako mazadlo otázek, ale tentokrát kód ankety na identifikaci nestačí, proto si pro proměnnou $_GET['smazodpoved'] připravíme id odpovědi.

Tlačítko "Ulož změny" je tu proto, aby bylo čím odeslat hodnoty z textových polí. Kliknutí na nějaký mazací odkaz formulář neodešle, na to pozor.

A nakonec vyhodnocení. Setkáme se tu s něčím, s čím jsme se ještě nesetkali: s neznámým počtem vstupních parametrů, u kterých ani přesně nevíme, jak se jmenují. Začnu kódem a potom si to vysvětlíme:

```
if (($vybranaanketa<>'')
{

//úprava otázky:
if(isset($_POST['editotazka']))
mysql_query("update otazky set otazka='".$_POST['editotazka']."' where kodankety=$vybranaanketa",$spojeni);

//funkce určující, jestli daná věc je jméno proměnné s odpovědí:
function jetoodpoved($co)
{
return(substr($co,0,11)=='editodpoved');
}

//úprava odpovědí:
$klice=array_keys($_POST);
$klice=array_filter($klice,'jetoodpoved');
foreach ($klice as $klic)
{
$idecko=substr($klic,11);
mysql_query("update odpovedi set odpoved='".$_POST[$klic]."' where id=$idecko",$spojeni);
};

//případné smazání ankety (tj. otázky a všech odpovědí, které k ní patří):
if(isset($_GET['smazanketu']))
{
mysql_query("delete from odpovedi where kodankety=$vybranaanketa",$spojeni);
mysql_query("delete from otazky where kodankety=$vybranaanketa",$spojeni);
$vybranaanketa=''; //to aby bylo jasné, že tahle anketa už neexistuje
};

//případné smazání některé odpovědi:
if(isset($_GET['smazodpoved']))
mysql_query("delete from odpovedi where id=".$_GET['smazodpoved'],$spojeni);

};
```

Uf. Takže co jsme to vlastně udělali?

1. Úprava otázky - celkem triviální věc. Pokud nám příslušná proměnná z Postu přišla (tj. pokud jsme se sem dostali odesláním formuláře), nahradíme její hodnotou hodnotu v databázi otázek. O jednoznačnou identifikaci se postará kód vybrané ankety. Úspěšnost už ani netestuju - buď se to povede, nebo se nám ve formuláři objeví zase ta původní hodnota a pak je jasné, že se to nepovedlo. Ale prakticky se to stejně vždycky povede.
2. Filtrovací funkce - řekne nám, jestli daný řetězec začíná na 'editodpoved'. Funkce **substr**(Řetězec,Pozice,Délka) vrací úsek z daného Řetězce začínající od dané Pozice (počítáno od nuly) a o dané Délce, nebo pokud Délka není zadaná, tak bere všechno až do konce Řetězce. Filtrovací funkci budeme potřebovat o kousek dál.
3. Úprava odpovědí - tady už to trošičku houstne. Teoreticky by nebyl problém projít foreachem celé pole $_POST, u každé položky filtrovací funkcí (nebo přímo) otestovat klíč jestli je to jméno odpovědi, a jestli jo, její hodnotu uložit do tabulky. Ale já jsem se rozhodl pro zpestření použít trochu jiný postup. Funkcí **array_keys** jsem z pole $_POST vytáhl do samostatného pole klíče. Tohle nové pole je indexováno čísly od nuly a původní klíče má jako hodnoty. V dalším kroku jsem použil funkci **array_filter**, která z daného pole vyhodí všechny prvky, pro které funkce v druhém parametru (v našem případě "jetoodpoved", zadává se takhle jménem jako text) dává hodnotu false. Tím jsme dostali pole, které obsahuje pouze jména proměnných, které nás zajímají. Zbytek už je v podstatě triviální: každé toto jméno vezmeme, vytáhneme z něj id (víme, že začíná na pozici 11 a pokračuje až do konce jména), použijeme ho jako identifikační podmínku a přepíšeme text odpovědi.
4. Smazání ankety - nejdřív se podíváme, jestli přes Get přišla příslušná proměnná (tj. jestli jsme se sem dostali kliknutím na mazací odkaz). Pokud ano, smažeme všechny otázky a odpovědi, které patří k této anketě.
5. Smazání odpovědi - jednodušší než úpravy textů, protože smazat se dá vždycky jenom jedna, navíc její id dostaneme až pod nos. Myslím, že tady není co vysvětlovat.

A to je vše, přátelé!

Na to, jaká je to blbost, jsme si toho o anketách napsali až až . Doufám, že uvedené příklady aspoň k něčemu budou. Jestli dáváte přednost souvislým textům před rozkouskovaným seriálem, je vám jako obvykle k dispozici alternativní verze tohoto trojčlánku.

Příště se zaměříme víc na texty a napíšeme si (konečně) něco užitečnějšího: návštěvní knihu.

13. díl - Kniha návštěv - Ukládání textů do databáze

Vítejte u dalšího dílu seriálu o tvorbě dynamických doplňků na webové stránky (začátek je tady). Dnes si probereme ukládání textů do databáze, dokončíme zabezpečení SQL, nakousneme obranu proti spamovacím robotům a vytvoříme si svoji první návštěvní knihu.

Návštěvní kniha (anglicky guestbook) je stránka, na kterou můžou návštěvníci připisovat vzkazy. Provedení se může lišit od jednoduchého malého okénka (shoutboard) po plnohodnotné jednovláknové fórum s možností odpovídání, citací, přihlašování a podobně. V tomhle článku zůstaneme spíš na tom jednodušším konci spektra. Ale dost teorie, jde se na věc.

Datové struktury

Na ukládání vzkazů od návštěvníků nám postačí jedna tabulka, ve které bude každý řádek odpovídat jednomu vzkazu. Ukládat budeme:

- Vlastní text vzkazu.
- Identifikační kód, podle kterého budeme vzkazy řadit, hledat, mazat apod.
- Jméno autora.
- Datum (případně i čas) odeslání.
- Případné další věci jako třeba IP adresu odesilatele, mail, odkaz a podobně zatím pro přehlednost vynecháme.

Na text vzkazu nám asi nebude stačit TINYTEXT (255 znaků), ale TEXT postačí určitě - 64 KB nemá ani celý tenhle článek. Stejně by posloužil jakýkoli dostatečně dlouhý VARCHAR.

Na identifikační kód použijeme dostatečně velké přirozené číslo s automaticky nastavovanou hodnotou, jako obvykle. Jméno autora bude pravděpodobně celkem krátké, dlouhé traktáty tam píšou snad jen hloupí roboti. Dejme tomu, že 30 znaků by mohlo stačit. Teoreticky by se autoři mohli podepisovat přímo v textu vzkazu a oddělené políčko pro jméno by tedy nebylo nutné, ale nebylo by to moc praktické (půlka lidí by na to zapomínala).

Datum a případně i čas odeslání jsou docela užitečné údaje. Zároveň slouží i pro orientaci návštěvníka: podle nich bezpečně pozná, jakým směrem jsou příspěvky v knize řazeny (tedy pokud mu to nenaznačíme nějakým jiným způsobem, což doporučuji). Jako datový typ můžeme použít DATE nebo DATETIME, v našem příkladu si vystačíme s datem.

Tabulka by tedy mohla vypadat nějak takto:

```
CREATE TABLE nkniha
(
id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
datumvlozeni DATE,
autor VARCHAR(30),
obsah TEXT
)
```

Pozn.: čistě teoreticky by se kniha dala vytvořit i bez databáze, pouze zápisem do souboru. Ale to má několik háčků: zaprvé obtížné řazení od nejnovějšího příspěvku, zadruhé obtížné stránkování a zatřetí problémy, pokud přijde víc příspěvků současně. První dva háčky se dají obejít vytvořením samostatného souboru pro každý příspěvek (třeba s číselným jménem), se třetím nic nenaděláme.

Odesílání vzkazů

První základní funkce knihy. Potřebujeme na to odesílací formulář, do kterého nám návštěvníci budou svoje vzkazy psát. Stačit nám budou dvě textová políčka, datum a kód si doplníme sami:

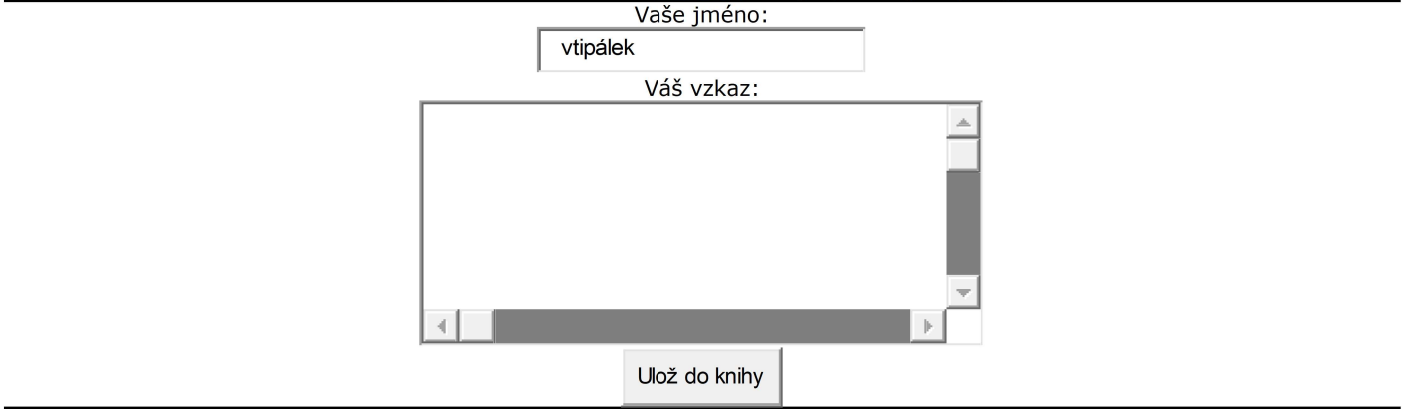
```
<form action="nkniha.php" method="post">
Vaše jméno:<br>
<input type="text" name="jmeno" maxlength="30" size="30"><br>
Váš vzkaz:<br>
<textarea name="vzkaz" rows="7" cols="50" wrap="soft"></textarea><br>
<input type="submit" value="Ulož do knihy">
</form>
```

Po odeslání formuláře se zadaný vzkaz předá skriptu **nkniha.php**. Asi nejpraktičtější bude, když dáme všechno - formulář, zpracování i zobrazení - do jednoho skriptu (pak je atribut action zbytečný), ale teoreticky by to šlo zařídit i odděleně. Skript nkniha.php tedy dostal návštěvníkův vzkaz a měl by ho uložit do databáze. Tady máte první nástřel - schválně zkuste přijít na to, co je na něm špatně:

```
mysql_query("INSERT INTO nkniha VALUES (NULL, CURDATE(), '".$_POST['jmeno']."', '".$_POST['vzkaz']."')",$spojeni));
```

Příkaz INSERT, čili vložení řádku do tabulky, už dávno známe. Hodnota NULL na místě identifikačního kódu je v pořádku - nic jiného by tam ani nešlo, dosazení správného čísla zajistí auto_increment. Funkce SQL **CURDATE()** je novinka: dá nám aktuální datum přesně v takovém tvaru, jaký vyžaduje datový typ DATE. Zadané jméno a vzkaz, které se uloží do sloupců autor a obsah, jsou syntakticky správně. Parametr $spojeni je jako obvykle výstup z mysql_connect, na tom není co zkazit. Takže?

Jasně, bezpečnost. Kdo dával minule pozor, ví, co to je SQL injekce:



Po dosazení zadaných textů do výše uvedeného příkazu by databáze dostala tohle:

```
INSERT INTO nkniha VALUES (NULL, CURDATE(), 'vtipálek', 'nic'); DROP TABLE nkniha; --')
```

Ano, totéž jako známý vtip z xkcd. Apostrof za slovem "nic" ukončil textový parametr, závorka a středník ukončují příkaz INSERT, následuje nový příkaz a nakonec dvě pomlčky zakomentují případný zbytek původního příkazu, aby ho SQL nebralo jako syntaktickou chybu. Jak prosté, milý Watsone...

V našem případě to naštěstí není tak horké, funkce mysql_query totiž dokáže zpracovat pouze jeden příkaz. Když jich dostane víc (i když jsou syntakticky v pořádku a oddělené středníkem), neprovede ani jeden a rovnou vrátí false. Ale byla by chyba se na to spoléhat - co kdyby to nějaká verze dělala jinak, že jo.

Uživatelské texty vkládané do příkazů proto musíme **vždy** prohnat nějakou escapovací funkcí, která zajistí, že se z textu stane opravdu jenom neškodný text, který SQL za žádných okolností nevyhodnotí jako příkaz. První možností je standardní phpčkovská funkce **addslashes**, která ale používá neměnný algoritmus a nebere v úvahu případné syntaktické odlišnosti konkrétní databáze. Nejjistější je **mysql_real_escape_string**, která je stavěná databázím SQL na míru:

```
$zabezpeceny_text=mysql_real_escape_string($puvodni_text,$spojeni);
```

Funkce před každý apostrof a další řídicí znaky vloží znak "\". Interpret si po vyhodnocení příkazu lomítka vyhází, takže v databázi už bude text uložený bez nich, přesně tak, jak ho uživatel napsal.

Mysql_real_escape_string je strašně dlouhé slovo, z hlediska pohodlí se vyplatí zabalit si ho do nějaké funkce s kratším jménem. Já například používám tohle:

```
function zabezpec($retezec)
{
global $spojeni;
return mysql_real_escape_string($retezec,$spojeni);
}
```

Novinkou je pro nás slovíčko **global**, které říká, že proměnná $spojeni je globální. To je nutné, protože **v PHP se všechny proměnné definované uvnitř funkcí berou jako lokální**. Kdyby tam ten řádek nebyl, mysql_real_escape_string by skončila chybou, protože by v omylem vytvořené lokální proměnné $spojeni pochopitelně nenašla platné spojení s databází.

Takže tedy ukládání příspěvků do knihy, znovu a lépe:

```
mysql_query("INSERT INTO nkniha VALUES (NULL, CURDATE(), '"
.zabezpec($_POST['jmeno'])."', '"
.zabezpec($_POST['vzkaz'])."')",$spojeni));
```

Teď už je to v pořádku, tohle nám nikdo nehackne.

Kdy zabezpečení provádět, to už je na vás. Buď až těsně před použitím jako v tomto příkladě, nebo o něco dříve, nebo si třeba můžeme hned na začátku pro jistotu zabezpečit rovnou celý POST a máme po starostech:

```
foreach($_POST as $klic=>$hodnota)
$zpost[$klic]=mysql_real_escape_string($hodnota,$spojeni);
```

Pole **$zpost** možná budu používat i v dalším textu. Vždycky bude představovat zabezpečenou kopii $_POST a nebudu zbytečně opakovat, jak jsme se k ní dostali. Obdobný význam bude mít pole **$zget** pro $_GET.

Dobrá, proti SQL injekci jsme obrnění. Zbývá ještě nepodstatná drobnost zvaná doubleposty, čili **duplicitní příspěvky**. Ty vznikají, když někdo po odeslání vzkazu během netrpělivého čekání na odezvu serveru zmáčkne F5 - tím se data z formuláře odešlou znovu (některé prohlížeče se ptají, některé odesílají rovnou). Zabráníme tomu tak, že ihned po uložení příspěvku skript restartujeme, ale tentokrát bez formulářových dat. Jak? Skokem pomocí funkce header:

```
header('Location: nkniha.php');
```

Nebo kratší a blbuvzdornější forma, která říká "skoč na ten skript, ve kterém zrovna jsi" :

```
header('Location: .');
```

To samozřejmě předpokládá, že zpracování nového příspěvku je to první, co náš skript dělá, a nic během toho nevypisuje na stránku, jinak by header nefungoval. Ale to už znáte. Další samozřejmá věc je, že header voláme z vnitřku podmínky "pokud byl zadán nějaký text", jinak by se nám skript zacyklil donekonečna.

14. díl - Kniha návštěv - Zobrazování textů z databáze

Zobrazování příspěvků je druhá základní funkce návštěvní knihy - bez ní by to nebyla kniha, ale černá díra.

První otázka je, v jakém pořadí chceme vzkazy ukazovat: buď odshora dolů od nejstaršího k nejnovějšímu (což se líp čte, proto se to používá na fórech nebo tam, kde se očekávají hodně dlouhé příspěvky), nebo obráceně (takže bez zdlouhavého rolování rovnou vidíte nejnovější konec diskuse). Dejme tomu, že vzkazy v naší knize budeme řadit od nejnovějšího. Kód pro zobrazení všech vzkazů najednou by mohl vypadat třeba takhle:

```
$vysledek=mysql_query("SELECT datumvlozeni, autor, obsah FROM nkniha ORDER BY id DESC",$spojeni);
while ($radek=mysql_fetch_array($vysledek)):
echo '<p><b>'.$radek['autor'].'</b> ';
echo '('.$radek['datumvlozeni'].')<br>';
echo $radek['obsah'].'</p>';
endwhile;
```

To je asi nejjednodušší možná varianta. Má jeden podstatný nedostatek: texty sypeme do HTML kódu stránky tak, jak je uživatelé napsali, a prohlížeči, snaž se. Můžete si být jisti, že se brzy najdou vtipálci, kteří vám knihu zaneřádí v lepším případě blikajícími obrázky přes celou obrazovku (stačí tag <img>) a odkazy na stránky prodávající zaručeně pravé hodinky a zázračné pilulky, v horším případě pak klientskými skripty a aktivními objekty, které si s počítačem návštěvníka mohou dělat, co chtějí. Takže tudy ne, přátelé. Chce to text projít a potenciálně nebezpečné tagy zneškodnit.

První možnost je funkce **strip_tags**. Ta ze zadaného textu veškeré HTML tagy (včetně komentářů a úseků <?php ... ?>) prostě vymaže. Kdyby se vám to nehodilo, můžete ještě doplnit druhý parametr se seznamem tagů, které se mají zachovat, např.:

```
$novy_text=strip_tags($puvodni_text,'<br><p><img>');
```

V takovém případě ale pozor, že u zachovaných tagů zůstanou všechny jejich atributy, i ty potenciálně nebezpečné (onload, onmouseover apod.).

No jo, ale co když nám do knihy někdo bude chtít napsat, že a<b nebo že nemá rád tag <font>? V takovém případě by mu strip_tags smazala půl příspěvku. Naštěstí existuje druhá možnost, funkce **htmlspecialchars**. Ta aktivní znaky jako <, ", & a podobně přepíše na HTML entity (< " & atd.), takže je potom prohlížeč jednak zobrazí tak, jak byly napsány, a jednak je nebude vyhodnocovat jako HTML kód, takže nám nehrozí žádné skriptové útoky.

Nebezpečného HTML kódu jsme se tedy zbavili, ovšem pisatelé tím přišli o veškeré možnosti formátování, včetně tak základní věci, jako je zalamování řádků. Enter napsaný do textového pole sice vloží do textu znak \n (nový řádek), ale ten se v HTML projeví jenom jako mezera. Naštěstí i na tohle máme chytrou funkci, **nl2br**, která před všechny znaky \n vloží tagy
. Kdybyste chtěli klasický tvar
, musíte přidat druhý parametr s hodnotou false:

```
$novy_text=nl2br($puvodni_text,false);
```
Při true nebo při vynechaném druhém parametru se vkládá varianta s lomítkem.
To je sice docela hezké, ale poněkud jednoúčelové. Co kdybychom chtěli nahrazovat úseky textu nějak obecněji? Třeba textového smajlíka ":-)" grafickým "<img src="smajlik1.gif">"? S tím si poradí funkce **str_replace**(co,čím,kde):
```
$novy_text=str_replace(':-)','<img src="smajlik1.gif">',$puvodni_text);
```
Str_replace je velice užitečná věc, která určitě najde uplatnění i v mnoha jiných aplikacích.
Poslední věc, která by se nám mohla nelíbit, je datum. Pokud ho zobrazíme tak, jak nám přijde z databáze, bude vypadat nějak jako 2012-08-10. To je dobré leda tak pro Američany, my bychom radši 10. 8. 2012. Na formátování data má SQL funkci **DATE_FORMAT**, pro uvedený tvar data by vypadala takhle:
```
SELECT DATE_FORMAT(datumvlozeni,'%e. %c. %Y') FROM nkniha
```
První parametr je datum ke zformátování, druhý je požadovaný tvar. Každý kód %+písmeno má nějaký význam (třeba "číslo dne", "číslo měsíce bez úvodní nuly" apod.), kompletní seznam najdete v manuálu. Ostatní znaky (např. ty tečky a mezery) se zobrazí tak, jak jsou. Výsledkem funkce je textový řetězec.
No jo, jenže jak se ten výsledek bude jmenovat, čili jaký index máme použít ve funkcích mysql_fetch_array nebo assoc? Upřímně řečeno, nevím a ani to vědět nepotřebuju. V SQL se totiž dá použít alias, tj. předefinovat jméno libovolného sloupce. Dělá se to klíčovým slovem **AS** (česky "jako") a v našem případě by to mohlo vypadat třeba takhle:
```
$vysledek=mysql_query("SELECT DATE_FORMAT(datumvlozeni,'%e. %c. %Y') AS upravenedatum, autor, obsah FROM nkniha ORDER BY id DESC",$spojeni);
$radek=mysql_fetch_array($vysledek);
echo $radek['upravenedatum'];
```
Aliasy jsou docela užitečná věcička - jak u takovýchto funkcí s blíže neurčeným názvem výsledku, tak např. u výběru dat z více tabulek současně, kde se může sejít několik sloupců se stejným jménem (typicky třeba id) a je potřeba je nějak odlišit. Ale o tom si povíme jindy.
Ale dost už vylepšování textů, nebo nám z nich nic nezbyde. Další důležitou otázkou je, kolik příspěvků najednou chceme ukázat. Vypisovat všechny by nebylo moc praktické - čím víc jich bude, tím déle by se kniha načítala a v půl kilometru vysoké stránce by se stejně nikdo nevyznal. Takže to raději nějak omezíme, nejlépe stránkováním po určitém počtu vzkazů, a přidáme nějaké ovládací prvky, kterými si návštěvník může nalistovat všechno, co bude chtít.

Stránkování zobrazených příspěvků

Předpokládám, že na další stránky nás dostane nějaký odkaz "zobraz starší/novější vzkazy", čili nějaká proměnná předaná metodou GET (POSTový formulář s jedním tlačítkem by teoreticky šel taky, ale neprošli by přes něj vyhledávací roboti). Na určení polohy by se dala použít přímo položka id, ale předpokládám, že vzkazy budeme občas i mazat, takže některé hodnoty id přestanou existovat a byly by z toho potíže. Takže si zavedeme hodnotu "číslo stránky". Dejme tomu, že deset nejnovějších vzkazů bude na stránce 0, dalších 10 bude na stránce 1 atd., až k tomu úplně nejstaršímu. V SQL se výběr provede omezením Selectu pomocí klíčového slova LIMIT:
```
SELECT * FROM tabulka LIMIT x,n
```
Tím se z tabulky vybere n řádků, počínaje od xtého (x se počítá od nuly). Dá se to kombinovat i s dalšími omezujícími podmínkami, jako třeba WHERE (v takovém případě Limit ořezává až to, co po Where zbyde). V našem konkrétním případě tedy může příkaz vypadat např. takto:
```
$vysledek=mysql_query("SELECT * FROM nkniha ORDER BY id DESC LIMIT ".(10)*$stranka.",10",$spojeni);
```
Na stránce 0 se vybere deset nejnovějších. Na stránce 1 se těch deset nejnovějších přeskočí a vybere se dalších 10 od jedenáctého dál a tak dále. Číslo 10 je v závorce proto, že tečka jako operátor zřetězení má hodně malou prioritu, takže by si ji PHP vyložilo jako desetinnou tečku (tedy číslo 0.10) a skript by nefungoval. V praxi je každopádně výhodnější místo čísla použít proměnnou, abychom kvůli každé změně délky stránky nemuseli upravovat čísla rozházená po celém zdrojáku.
Pozn.: Možná vás napadlo, že bychom stejně dobře mohli z databáze vzít všechno a výsledky si probrat až v PHP v nějakém cyklu. Ano, teoreticky by to samozřejmě šlo, ale bylo by to výrazně pomalejší a náročnější na paměť. Zatímco SQL stačí jednou si přečíst zadaný příkaz, sáhnout do databáze a za pár milisekund vám vrátí požadovaný očesaný výběr, PHP musí v každém průchodu očesávacím cyklem znovu a znovu interpretovat každý řádek. V PHP obecně platí, že je vždycky výhodnější používat standardní funkce nebo pečlivě nakombinované databázové příkazy, než skládat složité algoritmy a cykly z jednoduchých příkazů.
Stránkovaný výběr tedy umíme, ale ještě se potřebujeme nějak dostat k hodnotě $stranka.
Dejme tomu, že výchozí stav bude stránka 0 - tu zobrazíme, pokud žádné číslo nepřijde nebo pokud místo čísla přijde nějaký nesmysl (ať už neúmyslný překlep nebo pokus o SQL injekci). Pokud nějaké platné číslo přijde, použijeme ho. Třeba takhle:
```
if (isset($_GET['stranka'])          //je nějaké číslo zadané?
 and is_numeric($_GET['stranka'])  //a je to opravdu číslo?
  and ($_GET['stranka']>=0))       //a je nezáporné?
   $stranka=$_GET['stranka'];      //ano, použijeme ho
else $stranka=0;                   //ne, použijeme výchozí stránku
```
Jakmile známe aktuální stránku, můžeme sestavit stránkovací odkazy:
```
echo '<a href="nkniha.php?stranka='.$stranka-1.'">Novější příspěvky</a>';
echo '<a href="nkniha.php?stranka='.$stranka+1.'">Starší příspěvky</a>';
```
Samozřejmě bude potřeba umístit je na nějaké vhodné místo. Kam, to je věc názoru. Já osobně nemám rád, když mají podobu šipek doleva a doprava nebo slov "předchozí" a "další". Vzkazy v knize jsou poskládané svisle, tak proč by se mělo najednou listovat vodorovně? A slovem "předchozí" se myslí předchozí (dříve napsané, tedy starší) vzkazy, nebo předchozí (dříve navštívená, tedy novější) stránka? Ale to už je jenom kosmetický detail.
Docela příjemná vychytávka může být to, že odkaz nahoru skočí na spodní konec novější stránky (odkazem s #kotvou), což čtenářům ušetří mačkání Endu (stejně by četli odspoda). Také není od věci odkaz na novější stránku nezobrazovat, pokud jsme na stránce 0. Obdobně můžeme naopak nezobrazit odesílací formulář, pokud na stránce 0 nejsme. O něco těžší by bylo nezobrazit odkaz na starší stránku, pokud jsme úplně na začátku knihy. Dále můžeme přidat odkazy úplně na začátek a úplně na konec. Trefit se na stránku 0 je triviální, číslo poslední stránky by se vypočítalo z celkového počtu vzkazů v knize. Ten by se zjistil takto:
```
SELECT COUNT(*) FROM nkniha
```
Příkaz nám vrátí jednoprvkovou návratovou tabulku, ze které si příslušné číslo vytáhneme třeba přes mysql_fetch_row.
Nyní tedy víme, jak ukládat vzkazy do databáze a jak je z ní vybírat a zobrazovat. Zbývá poslední kapitola: obrana proti spambotům.

15. díl - Kniha návštěv - Obrana proti komentářovému spamu

Cílem spammera je rozšířit po internetu co nejvíc odkazů na určité stránky. Vyhledávače si pak myslí, že když na ně vede tolik odkazů z tolika míst, jsou asi fakt dobré, a tak je ve výsledcích zobrazují na přednějších místech.

Většina spammerů to samozřejmě nedělá ručně, ale napíší si na to robota - program, který automaticky prochází internet, hledá slibně vypadající formuláře a zkouší do nich psát. Co jsem tak vypozoroval, spamboti se na našich stránkách chovají zhruba takhle:

1. Nejdřív se musí zorientovat a najít návštěvní knihu. Orientaci jim velice usnadňují odkazy s texty jako "Guestbook", "Discussion board", "Forum" a podobně. První obrannou taktikou tedy je pojmenovat odkazy nějak jinak - například česky, tomu roboti obvykle nerozumí. Když už potřebujete angličtinu, docela pomáhá, když nahradíte některá písmena ASCII kódy (třeba e místo e) - lidé to přečtou bez problémů, ale pro robota je to úplně jiný řetězec.
2. Nejhloupější roboti hledání vzdají, ale ostatní dříve nebo později formulář knihy najdou. Nejdřív se podívají, jestli vypadá jako návštěvní kniha. Má aspoň jedno velké textové pole (textarea)? Má odesílací tlačítko, na kterém je v ideálním případě napsáno něco jako "Submit", "Post", "Save" nebo "Send"? Jsou tam ještě nějaká další textová políčka, nejlépe s předvyplněnými hodnotami jako "@", "http://" a tak? Jestli to vypadá slibně, robot do formuláře nasype svoje naprogramované bláboly a odešle je. V té chvíli musí zasáhnout naše druhá a nejdůležitější obranná linie, která rozezná spam od smysluplného příspěvku a zahodí ho. Metody rozpoznávání spamu si podrobněji probereme za chvilku.
3. Po chvilce si robot stránku znovu načte a podívá se, jak jeho první pokus dopadl. Jestli tam svůj výplod najde, zajásá a začne knihu stejným nebo podobným textem bombardovat, dokud ji úplně nezamoří. Pokud ho ale nenajde, usoudí, že to buď návštěvní kniha není nebo že je nějak moc dobře zabezpečená a odtáhne s nepořízenou (proto se třeba moc nespamuje přes mailovací formuláře). Jestli jsme robota napoprvé neprokoukli a něco už nám do knihy napsal, můžeme ho teď aspoň odlákat tím, že nové příspěvky zobrazíme až po nějakém čase (třeba po hodině nebo po ručním schválení) nebo na jiné stránce. Otázka je, jak moc to pomůže proti spambotům a jestli to spíš neodradí návštěvníky.
4. Stránky, kde se spamování daří, si roboti určitě dobře zapamatují a budou se tam rádi vracet.

Jak odhalit robota?

I druhou obrannou linii můžeme rozškatulkovat do několika vrstev. V první se snažíme roboty nachytat: zkoumáme, jakým způsobem odeslali formulář nebo kde v něm udělali nějakou botu. Sem patří různé kontrolní otázky, maskované ovládací prvky a podobně. Druhá vrstva zkoumá samotný text vzkazu a snaží se uhádnout, jestli je to spam nebo ne. To ovšem vyžaduje určitý stupeň umělé inteligence a rozsáhlou databázi vzorků, což přesahuje jak úroveň tohoto seriálu, tak úroveň mých znalostí 🙂. Třetí vrstva jsou různé závěrečné nouzové pojistky (jako třeba omezení počtu příspěvků), které zabrání totálnímu zahlcení knihy v případě, kdyby nějaký chytřejší robot prošel až sem.

Několik různých technik přechytračení robotů si teď probereme podrobněji.

Neviditelné prvky formuláře

Roboti vidí stránku jako HTML zdroják a je jim celkem jedno, jak vypadá v prohlížeči. Nejspíš je ani nenapadne zkoumat všechny přidružené definice CSS, jestli náhodou někde něco nemá vlastnost "display:none". To znamená, že na roboty můžeme nastražit past: pokud něco napíšou do neviditelného textového políčka nebo kliknou na neviditelné tlačítko, máme je - člověk to udělat nemůže.

```
<html>
<head>
<style type="text/css">
.viditelna {display:block}
.neviditelna {display:none}
</style>
</head>
<body>
<form method="post">
<input type="text" name="jmeno" class="viditelna">
<input type="text" name="past1" value="http://" class="neviditelna">
<textarea name="vzkaz" class="viditelna"></textarea>
<input type="submit" name="past2" value="Submit" class="neviditelna">
<input type="submit" name="odeslani" value="Uložit" class="viditelna">
</form>
...
<?php
if ( isset($_POST['past1']) and ($_POST['past1']<>'http://')
or isset($_POST['past2']) )
...je to robot...
else
...je to člověk...
?>
```

(Vyhodnocení dat jsem pro přehlednost dal až za hlavičku HTML a formulář, v praxi by bylo zřejmě úplně na začátku, aby se v případě potřeby dala použít funkce header.)

Výhoda téhle taktiky je jasná: neotravuje návštěvníky, protože si jí vůbec nevšimnou. Navíc je velice účinná, obzvláště když alternativních odesílacích tlačítek vytvoříte víc a jen jedno bude správné, případně pokud je ještě navíc náhodně střídáte.

Nevýhody? Pokud se sem spammer podívá ručně, prokoukne to hned, svého robota upraví a má vystaráno. Někteří roboti se možná v CSS vyznají a stylu display:none si všimnou. Také nevím, jak by to fungovalo v čistě textových prohlížečích s hlasovým výstupem pro slepce (v nejhorším tam můžeme dát popisky jako "sem nic nepište a na tohle neklikejte, nebo vám to smažu").

Matení Javascriptem

Tak, jako se dá pomocí CSS text skrýt, dá se Javascriptem napsat nebo různě měnit. Se správnou interpretací JS mají občas problémy i některé starší prohlížeče, natožpak roboti. Takže pokud si například formulář zobrazíme pomocí document.write() nebo si vhodným způsobem pohrajeme s událostmi onclick, onsubmit a onchange, zamotají se v tom.

Výhody jsou stejné jako prve: neviditelost pro lidi a účinné odpálkování robotů. Drobná nevýhoda: musíme spoléhat na to, že návštěvníci mají Javascript zapnutý a že jim v prohlížeči funguje tak, jak má. Dešifrování může být obtížné i pro živého spammera, ale jakmile ho prokoukne, napíše si parazitní formulář (tj. kopii našeho formuláře na svém počítači) a už se to s námi poveze.

Kontrola formátu zadaných dat

Tohle už trochu hraničí s inteligentními spamfiltry. Můžeme využít toho, že někteří boti píší dlouhé traktáty tam, kde je nečekáme (do políček pro jména, adresy a tak) nebo umisťují do textu spoustu odkazů (a href=...). Když se nám to nezdá, můžeme příspěvek zahodit. Délku textového řetězce zjistíme pomocí funkce **strlen**:

```
if (strlen($_POST['jmeno'])>50)
  ...podezřelé!...
else
  ...asi v pořádku...
```

Výhoda: další nenápadná vrstva. Nevýhody: malá účinnost (roboti už většinou tak blbí nejsou) a možnost nechtěného smazání příspěvku od člověka (nevěřili byste, jaké hrůzy si v návštěvních knihách vyměňují třeba programátoři).

Kontrola přes session nebo cookie

Tohle nemám ověřené, berte to spíš jako brainstorming. Skript, který zobrazuje formulář knihy, by mohl zároveň do session nebo cookie vložit nějakou proměnnou a při vyhodnocování odeslaných dat zkontrolovat její přítomnost. Nevím, jestli to má vůbec nějaký smysl, ale teoreticky by to aspoň mělo ztížit použití parazitních formulářů.

Turingův test neboli Captcha

Jde o různé kontrolní otázky, opisování pokrouceného textu z obrázků a podobné věci, které by člověk teoreticky měl zvládnout levou zadní, ale roboti by si na nich měli vylámat zuby. Účinnost je zpravidla poměrně slušná, ale hodně záleží na tom, jaký test zvolíme a jak je který robot vybavený. Zásadní nevýhoda je, že testy otravují především lidi a občas jim dělají větší potíže než robotům (nečitelné obrázky apod.). Další mínus je omezená použitelnost pro slepce a textové prohlížeče.

Různých kontrolních testů existuje nepřeberné množství, tady uvedu jenom pár příkladů, se kterými jsem se dosud setkal:

- Opisování náhodných textů z obrázku. V současnosti jedna z nejrozšířenějších metod. Výhodou je, že stačí připravit algoritmus a potom už to běží samo. Podrobný návod tu právě vychází.
- Několik zatrhávacích políček, kterými se hodnotí pravdivost výroků jako třeba "jsem člověk", "přišel jsem sem spamovat", "umím číst" a podobně. Kontrolní věty je potřeba předem připravit a čas od času obměňovat, ale zase je výhoda, že jde o ryze textový a dobře čitelný systém.
- Otázka, na kterou je potřeba odpovědět slovy. Nevýhoda je (kromě nutnosti předem vymyslet otázky) možná nejednoznačnost odpovědi. Třeba na otázku "V čem plavou ryby?" existuje spousta možných odpovědí: ve vodě, v rybníce, v moři, v akvárku atd., plus hromada jinak skloňovaných tvarů, s diakritikou nebo bez.
- Přepis foneticky zapsaného čísla (např. "stopjetašedesát") do číselného tvaru (165). Chce to předem si připravit čísla (kdyby šla vygenerovat náhodně, nebylo by luštění pro roboty takový problém), ale jinak je to relativně praktická a pohodlná metoda.
- Několik odesílacích tlačítek a opodál poznámka typu "příspěvek uložíte kliknutím na tlačítko 2", "...kliknutím na tlačítko s nejdelším textem", "...kliknutím na tlačítko úplně nalevo" a podobně. Tady si musíte předem připravit jak texty, tak tlačítka. Ale jinak je to celkem výhodná metoda, protože člověk nemusí zbytečně klikat ani psát nic navíc, jenom se rozhodne, kam ten jeden závěrečný klik umístí.
- Něco podobného, jenom máme místo odesílacích tlačítek (input type="submit") obrázek (input type="image") a je potřeba kliknout do něj na určité místo (třeba "příspěvek uložíte kliknutím na žirafu"). Souřadnice se předají spolu s ostatními daty formuláře jako proměnné $_POST['x'] a $_POST['y'], počátek 0,0 je v levém horním rohu obrázku. Výhody stejné jako minule, nevýhodou je nečitelnost pro slepce a větší náročnost na místo (potřebujeme spoustu předpřipravených obrázků).
- Spočítání jednoduché matematické úlohy, třeba 1+3=__, 19-4=__ a podobně. Čísla i operátory se dají generovat náhodně, takže odpadá nutnost něco si předem připravovat. Ovšem naučit robota počítat je triviální věc a jakmile se sem jednou podívá živý spammer, máme po ptákách.
- Ověření přes mail. Návštěvník do formuláře vyplní svoji mailovou adresu, na tu mu pošleme nějaký odkaz nebo heslo, se kterým něco udělá a až tím dojde k potvrzení odeslaných dat. Velice bezpečné, ale jinak naprosto extrémní pruda použitelná jedině při jednorázových úkonech (třeba registracích), ne pro každodenní psaní do diskusí.

Ale počkat... náhodně vybraný obrázek nebo otázka je hezká věc, ale jak náš skript při vyhodnocování odpovědi pozná, kterou otázku předtím vybral?

První a nejtriviálnější možnost je otázku nějak označit přímo ve formuláři, třeba skrytým prvkem (input type="hidden") s nějakým kódem otázky. Vyhodnocovací algoritmus to potom má jednoduché: patří k tomuhle kódu tahle odpověď? Ano/ne, hotovo. Nevýhoda je v tom, že jakmile se živý spammer mrkne do zdrojáku, vykopíruje si kód otázky do parazitního formuláře, přihodí svoji odpověď a rozjede to ve velkém.

Lepší je kód otázky nějak zamaskovat. Buď tím, že se pošle jinudy než přes formulář (třeba přes session nebo cookie), nebo že se zahashuje, třeba funkcemi md5(), crypt() nebo hash(). Ideálně třeba v kombinaci s aktuálním datem - nikoho nebude bavit každý den ručně krmit robota novým kódem. Nejlepší bude ukázat si to na příkladu. Dejme tomu, že máme otázky a odpovědi uložené v polích $otazky a $odpovedi s indexy od 1 do 100.

```
echo '<form method="post">';
$cislootazky=rand(1,100); //náhodný výběr otázky
echo $otazky[$cislootazky];
echo '<input type="text" name="odpoved">';
echo '<input type="hidden" name="kontrolnihash" value="'.md5($odpovedi[$cislootazky].date('Ymd')).'">';
echo '<input type="submit" value="Odeslat"></form>';
```

A teď vyhodnocení:

```
if (md5($_POST['odpoved'].date('Ymd'))==$_POST['kontrolnihash'])
  ...v pořádku...
else
  ...špatná odpověď...
```

Živý spammer sice ze zdrojáku formuláře zjistí, jaký hash odpovídá dnešnímu datu a téhle odpovědi, ale samotné datum ani odpověď z něj nevykouká.

Jo, a máme tu pár nových funkcí: **md5** vytvoří z daného řetězce hash (32znakový text z písmen '0'..'f'), **rand(a,b)** dává náhodné celé číslo z rozsahu a..b a **date** dává aktuální datum zformátované do daného tvaru (viz manuál). Kdyby nám šlo o bezpečnost (např. šifrování hesel), je lepší místo md5 použít crypt.

To by k captchám zatím mohlo stačit, teď si ještě probereme posledních pár taktik spamové války:

Omezení kadence příspěvků

Tohle už patří do kategorie "poslední záchrana, když všechny předchozí metody selžou". Jde o omezení počtu příspěvků, které se dají odeslat v určitém časovém intervalu (třeba během jednoho dne) z jednoho počítače. Prakticky jediná použitelná identifikace

počítačů jsou v tomto případě IP adresy, protože na cookies se nám roboti nejspíš zvysoka vy-víte-co. Počítání přístupů a ukládání IP adres už jsme si nacvičili v předchozích dílech o počítadlech a anketách, takže by to pro vás neměl být problém. Pokud zjistíte překročení maximálního dovoleného počtu, přestanete příspěvky z téhle adresy ukládat a případně rovnou schováte i formulář. Ale počítejte s tím, že se takhle ubráníte jedině amatérům, co spamují ručně z domova. Jakmile je někdo schovaný za proxy serverem, který mu IP každou chvíli mění, nebo ovládá velkou síť počítačů s různými adresami (botnet), ochranu vám pohodlně obejde.

Mazání duplicit

Další poslední záchrana. Normální člověk vám do knihy nenafláká deset navlas stejných vzkazů. Maximálně tak dva, když se uklikne, a i tak je ten druhý zbytečný. Takže můžeme každý nový příspěvek porovnat s několika předchozími a pokud se s některým shoduje, zahodit ho. Nevýhoda je, že roboti svoje bláboly většinou aspoň nepatrně obměňují.

Na závěr...

Nebojte se, není to tak horké. Spammeři si umí spočítat, kolik práce se jim ještě vyplatí vynaložit a kolik už ne. Nejvýhodnější je hacknout nějaký hodně rozšířený systém (jako třeba fórum PHPBB nebo návštěvní knihy od Blueboardu). Potom se ještě vyplatí zaspamovat nějaké velké a často navštěvované servery. Zbytek světa a amatérské stránky s jednotkami až desítkami návštěv za den už jim nestojí za individuální námahu. Raději si budou piplat svoje univerzální roboty, kteří zvládnou zaneřádit dostatečné množství dostatečně málo zajištěných stránek, ale na celý internet zatím nestačí.

A to je vše, přátelé!

Ještě bychom si mohli probrat mazání vzkazů z knihy, ale pro začátek si vystačíte s administračním rozhraním MySQL a časem vás určitě napadne, že se u každého příspěvku dá zobrazit nenápadný odkaz s jeho idčkem a nějak vhodně ho využít už by mělo být celkem jednoduché. Jestli ne, počkejte si na příští díl, kde si sestrojíme pořádné fórum s vlákny, registrovanými uživateli a administrátorským rozhraním.