Škola pre Mimoriadne Nadané Deti a Gymnázium, Teplická 7, 831 02 Bratislava

Anino BELAN

# SQL a PHP

učebný text pre septimu osemročného gymnázia

BRATISLAVA
2006

# Obsah

# Úvod

Tento kurz predstavuje stručný úvod do práce s databázami prostredníctvom jazyka SQL (Simple Query Language) a možnosti, ako vytvárať dynamické webové stránky s pomocou jazyka PHP (rekurzívna skratka z „PHP Hypertext Preprocessor“). Nadväzuje jednak na kurz tvorby HTML stránok, jednak na kurz jazyka C, ktoré ste mohli absolvovať u nás na škole a keď sa pustíte do jeho čítania, bolo by dobré aby ste ako o HTML, tak o jazyku C mali aspoň základnú predstavu. Jazyk PHP sa totiž vo svojich základných štruktúrach v mnohom zhoduje z jazykom C a je určený primárne na tvorbu webu. Časť týkajúca sa SQL jazyka je ale nezávislá od jedného aj od druhého a pokojne ju môžete študovať aj samostatne.

V texte aj v príkladoch som sa z priestorových dôvodov dopustil mnohých zjednodušení. Táto príručka nemá nahradiť používanie manuálov. Jej účelom je popísať základné mechanizmy, aby ste mohli začať používať databázy a/alebo tvoriť dynamické internetové stránky. Časom zistíte, že nejaké veci sa dajú spraviť lepšie, poriadnejšie a bezpečnejšie. Ale príklady by sa tak neúmerne zväčšili. Záujemcom o tému rozhodne odporúčam štúdium podrobnejšej literatúry.

Ďakujem kamarátke Alenke Murovej, ktorá sa databázami živí a bez ktorej rád a prispenia by nebola vznikla dvanásta kapitola tohto kurzu a od ktorej mám aj materiál, na základe ktorého vznikla druhá kapitola.

Anino

# 1. lekcia
# Výnimočne niečo iné, ako „Hello, world!“

Programátori sa potýkajú s rôznymi druhmi problémov. Môžu premýšľať nad tým, ako čo najrealistickejšie vykresliť mydlovú bublinku, môžu programovať algoritmy na zjednodušenie plynárenskej siete, môžu robiť mnohé iné veci. Ale aj tak najčastejšie robia čo môžu, aby udržali poriadok v nejakej obrovskej hromade dát a boli schopní z nej vytiahnuť údaje, ktoré od nich niekto požaduje.

Môže to pritom začať celkom nevinne. Niekto začne vyvíjať nejakú online hru. Najprv len pre svojich kamarátov. Jednoduché pravidlá, zopár skriptov, všetky dáta sa vojdú do troch súborov. V tých skriptoch síce treba trochu cvičiť s čítaním a zapisovaním do súborov, ale dá sa. A potom sa to začne rozrastať. Kamaráti povedia o jeho skvelom webe ďalším kamarátom, niekto navrhne nový druh tovaru, ďalší pridá nové medzigalaktické plavidlo, o chvíľu má zoznam hráčov 5000 položiek, každému treba registrovať jeho majetok, obsadené planéty, bojovú silu a obchodné kontakty. A skripty prestanú stíhať.

Podobne to môže vyzerať, keď niekto robí firemný softvér pre malú firmu skladajúcu počítače. Z malej firmy sa pomaly ale isto stáva stredná, treba evidovať tisíc druhov skladových položiek, desaťtisíc zákazníkov, faktúry, účtovníctvo a to, na čo kedysi stačil malý programček a dve excelové tabuľky začne byť problém udržiavať.

V oboch prípadoch je na mieste začať uvažovať o použití relačného databázového stroja.

Databázový stroj je program, v ktorom sa dajú vytvárať tabuľky a ktorý vie v tých tabuľkách rýchlo vyhľadávať odpovede na pomerne komplikované otázky. Pod slovom „tabuľka“ si ale netreba hneď predstavovať tabuľku, akú vytvárate v tabuľkovom kalkulátore (aj keď aj to je do istej miery databáza). Položkou v databázovej tabuľke môže byť článok alebo obrázok, teda aj objekty pomerne rozsiahle, ktoré by v tabuľke tabuľkového kalkulátora vyzerali zvláštne. Databázový stroj má mechanizmy na to, aby k dátam mohlo súčasne pristupovať viacero ľudí, takže programátor sa o to nemusí starať. Databázový stroj môže obsluhovať viacero databáz, z ktorých každá môže obsahovať množstvo tabuliek, ktorých údaje sa dajú rôzne kombinovať. Zmysluplnosť a do istej miery aj efektívnosť týchto tabuliek ale záleží od programátora a od projektu na ktorom pracuje. K téme efektívnosti sa vrátime neskôr.

S databázovými strojmi sa komunikuje jazykom, ktorý nesie hrdý názov SQL.[1] Databázové stroje, ktoré tento jazyk poznajú, sa vo všeobecnosti nazývajú SQL servery. Na naozaj veľké aplikácie sa najčastejšie používa drahý, ale skutočne veľmi rýchly softvér od firmy Oracle. Ďalší známy komerčný produkt je MS-SQL od firmy Microsoft. Z opensource scény sú najznámejšie servery PostgreSQL, ktorý je vhodnejší na náročnejšie aplikácie a MySQL, ktorý je jednoduchší a má tú výhodu, že je portovaný pod viacero operačných systémov včítane MS Windows, kde spolu s webserverom Apache a jazykom PHP môže tvoriť rovnako účinnú a efektívnu trojicu ako pod operačným systémom Linux.

1 Z anglického Simple Query Language – Jednoduchý otázkový jazyk.

My budeme v našich lekciách pracovať práve s databázou MySQL, aj keď veci, ktoré budeme vysvetľovať, fungujú podobne aj v ostatných databázach. Skúsenosti, ktoré získate by ste mali bez problémov vedieť použiť v ktorejkoľvek z nich.[2]

Každý databázový stroj má viacero rozhraní, ktoré programátorovi umožňujú robiť zmeny v tabuľkách a využívať výsledky otázok. Či už je to jednoduché konzolové rozhranie, alebo grafický správcovský nástroj, súbor funkcií v jazyku PHP, Java, C++ alebo v nejakom inom, alebo veľké tabuľkové šialenstvo nazývané MAGIC, ktoré sa používa na vývoj databázového softvéru. Začneme tradične – s konzolou a neskôr, keď sa naučíte niečo o jazyku PHP, povieme si niečo o tom, ako databázové funkcie používať v tomto jazyku.

Predstavte si, že databázový stroj beží na serveri `artus`. Pripojíme sa na neho s použitím konzolového klienta `mysql` nasledujúcim spôsobom:

```
mysql -h artus -u guest -p
```

Parametrom `-h` dávame klientovi vedieť, na akom počítači databázový server beží. V prípade, že je to ten istý počítač, ako ten, na ktorom spúšťame klienta, tento parameter uvádzať netreba. V našom prípade je meno počítača na ktorom server beží `artus`.

MySQL si ako nejaký samostatný operačný systém spravuje svojich používateľov. Parametrom `-u` mu dáme vedieť, kto vlastne sme. (Sme `guest`.) Ak sme pod OS Linux prihlásení pod rovnakým menom, ako je naše používateľské meno v databáze, tento parameter opäť nemusíme uviesť. Parametrom `-p` povieme serveru, že si od nás má vypýtať heslo. (Inak by klient predpokladal, že žiadne heslo nie je a server by ho do databázy nepustil.)

**Úloha č.1:** Prihláste sa na server.

Ak sa vám úspešne podarilo prihlásiť sa, na termináli uvidíte výzvu `mysql>` a môžete zadávať príkazy v jazyku SQL.

SQL príkaz môže mať viacero riadkov. Aby bolo jasné, že už skutočne skončil, treba napísať bodkočiarku, inak sa objaví znak `->` a očakáva sa, že v príkaze budete pokračovať.

Príkazy SQL jazyka sa snažia do istej miery tváriť, že je to normálna angličtina, čo má tú výhodu, že ak si neviete spomenúť, ako sa to vlastne písalo, často pomôže intuícia. Na začiatku si vyskúšame nejaké základné príkazy vhodné na zorientovanie sa.

Ak si napríklad chcete pozrieť, aké databázy server poskytuje, napíšete príkaz

```
SHOW databases;
```

Získate prehľad databáz, ktoré sú na serveri prístupné. V závislosti na tom, ako je server nakonfigurovaný, môžete medzi nimi uvidieť databázu `mysql`.[3] Táto databáza obsahuje systémové údaje, hovorí, kto má prístup k akej databáze a kto je vôbec oprávnený databázu používať.

---

2 Je pravda, že každá zo zmienených databáz má svoj vlastný dialekt jazyka SQL, ktorý sa jemne odlišuje od ostatných. Aj z tohto dôvodu bolo vyvinuté rozhranie ODBC, ktoré tvorí akýsi spoločný základ s pomocou ktorého sa dá pristupovať do každej zo zmienených databáz.

3 Ona sa na serveri nachádza vždy, ale nie každý používateľ ju musí nutne vidieť. Náš školský server je nastavený tak, že ju nevidíte.

Teraz je ten správny čas prepnúť sa do niektorej databázy. Robí sa to príkazom `USE`. Takže ak sa chcete prepnúť do databázy `mysql`, napíšete príkaze

```
USE mysql;
```

Server vám odpovie chybovou hláškou, ktorej súčasťou je obľúbené spojenie `Access denied`, ktoré naznačuje, že k danej databáze nemáte prístupové právo. Netreba si ale zúfať, boli pre vás vytvorené databázy, ku ktorým prístup máte. Každý z vás má vytvorenú databázu s názvom `guest_<vaše meno>`, ktorá je určená na vaše pokusy. Okrem týchto vašich databáz existuje jedna spoločná, ktorá sa volá `guest_common` a ktorá bude vaša „ostrá“ databáza. Budú v nej vaše spoločné dáta. Správajte sa k nim slušne a mažte ich iba v prípade nutnosti alebo po dohode s ostatnými.

**Úloha č.2:** Prepnite sa do databázy `guest_common`

Keď ste sa úspešne do databázy dostali, môžete sa pozrieť, aké tabuľky v nej vlastne sú. Schválne – skúste uhádnuť, akým príkazom sa to robí. Áno, hádate správne, je to `SHOW tables;` Zatiaľ by tam mala byť jediná tabuľka s názvom `klienti`. Môžete opäť hádať, akým príkazom primäť server, aby vám tabuľku `klienti` opísal. Ľudia vládnúci angličtinou opäť nemajú problémy uhádnuť. Príkaz znie `DESCRIBE klienti;` a vyprodukuje nasledujúci výstup (vyskúšajte!!!):

```
+----------------+------------------+------+-----+---------+----------------+
| Field          | Type             | Null | Key | Default | Extra          |
+----------------+------------------+------+-----+---------+----------------+
| klient_id      | int(10) unsigned |      | PRI | NULL    | auto_increment |
| meno           | char(15)         |      |     |         |                |
| priezvisko     | char(20)         |      |     |         |                |
| bydlisko_ulica | char(30)         |      |     |         |                |
| bydlisko_obec  | char(30)         |      |     |         |                |
| bydlisko_psc   | char(6)          |      |     |         |                |
| email          | char(25)         | YES  |     | NULL    |                |
| telefon        | char(20)         | YES  |     | NULL    |                |
| aktivny        | enum('Y','N')    |      |     | Y       |                |
+----------------+------------------+------+-----+---------+----------------+
9 rows in set (0.03 sec)
```

Každý riadok popisuje jeden stĺpec tabuľky `klienti`. Prvý riadok `klient_id` je istým spôsobom najzaujímavejší. V kolónke `Key` má uvedené `PRI`. Znamená to, že sa jedná o primárny kľúč. Takáto položka jednoznačne určuje záznam v tabuľke (a teda dva rôzne záznamy ju nemôžu mať rovnakú) a slúži na to, aby sa s pomocou nej rýchlo k nejakému záznamu dalo dostať.[4] V našom prípade sa jedná o číslo (to vidieť z kolónky `Type`), ktoré bude v našej databáze slúžiť ako evidenčné číslo jednotlivých klientov. V kolónke `Extra` má tento index uvedené `auto_increment`. To je vec, ktorá spôsobí, že keď sa bude do tabuľky pridávať nový klient, hodnotu `klient_id` vôbec nebude treba zadávať. Databáza si uloží najvyššiu dosiahnutú hodnotu a pri pridávaní ďalšej položky ju zväčší o 1 a pridelí.

Ďalšie riadky sa týkajú mena, priezviska a adresy prípadného klienta. Všetko sú to textové reťazce. Pri každom je v zátvorke určená jeho maximálna dĺžka. Dĺžku treba voliť tak, aby

4 Tabuľka môže mať viacero kľúčov podľa ktorých sa v nej dá rýchlo vyhľadávať. Primárny kľúč ale určuje spôsob, akým sa k tabuľke bude pravdepodobne pristupovať najčastejšie. O kľúčoch budeme podrobnejšie rozprávať neskôr.

bola dostatočná, ale aby položka nezaberala príliš veľa miesta. Jeden kilobajt prázdneho priestoru v jednej položke sa nezdá byť veľa, ale keď je položiek tisíc, je to už megabajt vyplytvaného priestoru.

Ďalšie dve položky (`email` a `telefon`) majú v kolónke `Null` uvedené `YES`. Znamená to, že môžu nadobúdať špeciálnu hodnotu `NULL`, ktorá znamená, že tam vlastne žiadna hodnota nie je. To pri predošlých položkách nebolo možné. (To ale neznamená, že tieto položky nemôžu obsahovať prázdny reťazec. Overenie toho, či sú údaje zadávané do databázy podľa vašej chuti je ale ponechané na softvér, ktorý do vašej databázy bude pristupovať.)

Posledná položka je typu `enum`. To znamená, že môže nadobúdať niektorú z vymenovaných možností. V našom prípade to je buď `Y` (ako Yes) alebo `N` (ako No). Týmto spôsobom sa bežne v databázach kódujú logické hodnoty.[5] Táto položka má v kolónke `Default` nastavené `Y` ako štandardnú hodnotu. Pri pridávaní nového klienta teda hodnotu `aktivny` netreba určovať, bude automaticky `Y`.

Keď sa chcete pozrieť, aké dáta sú v tabuľke uložené, slúži na to príkaz `SELECT`. Je to príkaz veľmi mocný s rôznymi podivnými možnosťami použitia. Jeho najjednoduchšia forma vyzerá takto:

```
SELECT * FROM klienti;
```

**Úloha č.3:** Vyskúšajte.

Posledná vec, ktorú si v tejto lekcii ukážeme je vkladanie nových údajov do tabuľky. Robí sa to (prekvapivo) príkazom `INSERT`. Príkaz `INSERT` má podobne ako príklad `SELECT` viacero podôb. Jedna z nich vyzerá takto:

```
INSERT INTO klienti
    ( meno, priezvisko,
      bydlisko_ulica, bydlisko_obec, bydlisko_psc,
      email )
VALUES
    ( "Jolana", "Laznibatová",
      "Teplická 7", "Bratislava", "831 02",
      "smnd@smnd.sk" );
```

Do zátvorky za menom tabuľky sme uviedli, ktoré položky v tabuľke budeme určovať. Ak by sme to nespravili, museli by sme určiť všetky hodnoty (a miesto hodnôt, ktoré sú určené automaticky alebo ich určiť nechceme písať `NULL`). Samotné hodnoty potom uvedieme do zátvorky za kľúčové slovo `VALUES`. Dávajte si pozor na to, že hodnoty polí typu `char` musia byť uvedené v úvodzovkách.

**Úloha č.4:** Vložte do tabuľky `klienti` aspoň 3 zmysluplne vyzerajúce položky. Vyskúšajte aj variantu príkazu `INSERT`, ktorá za menom databázy nemá určené, ktoré položky sa budú vkladať. S pomocou príkazu `SELECT` si prezrite vložené údaje. Všimnite si, aké hodnoty sú nastavené pri položkách, ktoré ste neurčili.

5 Samozrejme si tam pokojne môžete dať úplne iné písmená alebo položku definovať ako `tinyint` (malé celé číslo) a s jej hodnotou narábať ako s logickou hodnotou v jazyku C.

2. lekcia

# Ako robiť tabuľky alebo „Správny dizajn má svoje výhody“

Táto lekcia bude úplne teoretická. Povieme si, ako vytvárať v databáze tabuľky tak, aby sa zbytočne neplytvalo miestom, aby sa tam ľahko robili úpravy a aby sa v nich jednoducho zachovával aspoň aký-taký poriadok.

Okolo vhodného návrhu databáz je vytvorená skoro veda, ktorá hovorí, že aby bola databáza O.K., musí spĺňať niekoľko **normálnych foriem**. Väčšinu z nich by človek, ktorý sa nad vecou zamyslí vymyslel aj sám. Ale napriek tomu môže byť užitočné, keď tomu budeme venovať nasledujúcu lekciu.

Predstavme si hypotetického programátora Johnyho Nadšenca, ktorý ide vytvárať databázový systém pre obchod s ponožkami, ktorý si otvoril jeho kamarát Fero Fusakľa. Johny Nadšenec k tomu pristúpi spôsobom „napcháme to všetko do jednej tabuľky“. Kúsok jeho tabuľky môže vyzerať napríklad takto:

| Č.obj. | Dátum | ID Zák. | Meno | Adresa | ID Tovaru | Popis | Cena/ks | Počet | Cena | Cena/Obj. |
|---|---|---|---|---|---|---|---|---|---|---|
| 17 | 01.04.2005 | 341 | Dežo Točka | Kubická 3 | 17 | Zelené vlnené | 23,00 Sk | 2 | 46,00 Sk | 235,00 Sk |
| 17 | 01.04.2005 | 341 | Dežo Točka | Kubická 3 | 21 | Zamatové | 65,00 Sk | 1 | 65,00 Sk | 235,00 Sk |
| 17 | 01.04.2005 | 341 | Dežo Točka | Kubická 3 | 4 | Čierne bavl. | 15,00 Sk | 4 | 60,00 Sk | 235,00 Sk |
| 17 | 01.04.2005 | 341 | Dežo Točka | Kubická 3 | 8 | Ružové | 32,00 Sk | 2 | 64,00 Sk | 235,00 Sk |
| 18 | 02.04.2005 | 216 | Heda Divá | Kolmá 90 | 4 | Čierne bavl. | 15,00 Sk | 2 | 30,00 Sk | 193,00 Sk |
| 18 | 02.04.2005 | 216 | Heda Divá | Kolmá 90 | 21 | Zamatové | 65,00 Sk | 1 | 65,00 Sk | 193,00 Sk |
| 18 | 02.04.2005 | 216 | Heda Divá | Kolmá 90 | 9 | Krajkové | 49,00 Sk | 2 | 98,00 Sk | 193,00 Sk |

Nevýhody takto navrhnutej tabuľky sú zrejmé. V tabuľke sa nám hromadia zbytočné dáta. Dežo Točka je so svojou adresou a identifikačným číslom uvedený pri každom druhu ponožiek, ktoré si objednal. Druhá nevýhoda je, že vyhľadať v takejto tabuľke potrebné informácie je celkom problém. Nie je jasné, podľa akého kľúča by mala byť triedená. A zistiť z nej, koľko Fero zarobil na čiernych ponožkách, alebo ktorý zákazník u neho utratil najviac vyžaduje, aby bola utriedená podľa rôznych kritérií. (Nie že by samo o sebe bol problém tabuľku utriediť podľa viacerých kritérií, ale ak sa to preženie, databáza stratí prehľadnosť.)

Prvý krok bude dostať z tabuľky zbytočnú duplicitu. Spraví sa to jednoducho. Rozdelíme tabuľku na dve nasledujúcim spôsobom:

| Č.obj. | Dátum | ID Zák. | Meno | Adresa | Cena/Obj. |
|---|---|---|---|---|---|
| 17 | 01.04.2005 | 341 | Dežo Točka | Kubická 3 | 235,00 Sk |
| 18 | 02.04.2005 | 216 | Heda Divá | Kolmá 90 | 193,00 Sk |

| Č.obj. | ID Tovaru | Popis | Cena/ks | Počet | Cena |
|---|---|---|---|---|---|
| 17 | 17 | Zelené vlnené | 23,00 Sk | 2 | 46,00 Sk |
| 17 | 21 | Zamatové | 65,00 Sk | 1 | 65,00 Sk |
| 17 | 4 | Čierne bavl. | 15,00 Sk | 4 | 60,00 Sk |
| 17 | 8 | Ružové | 32,00 Sk | 2 | 64,00 Sk |
| 18 | 4 | Čierne bavl. | 15,00 Sk | 2 | 30,00 Sk |
| 18 | 21 | Zamatové | 65,00 Sk | 1 | 65,00 Sk |
| 18 | 9 | Krajkové | 49,00 Sk | 2 | 98,00 Sk |

V prvej tabuľke máme teraz záznam o objednávke. V tabuľke sa dobre dá vyhľadávať podľa čísla objednávky. Každé číslo objednávky sa v nej (na rozdiel od predošlej obrovskej tabuľky) nachádza len raz, takže môže slúžiť ako primárny kľúč. Údaje o zákazníkovi a dátume sa už zbytočne neopakujú.

Takže morálne poučenie je, že **ak k niečomu prislúcha viacero položiek[6], dajte to do samostatnej tabuľky**. Zrušíte tak zbytočné opakovanie, ušetríte tak miesto a zvýšite prehľadnosť. Keď ste svoju databázu upravili do tohto tvaru, dostali ste ju tak do **prvej normálnej formy**.

Dosiahli sme istý pokrok, ale stále to ešte nie je úplne optimálne. Každý riadok prvej tabuľky je jednoznačne určený údajom v prvom stĺpci – číslom objednávky. Druhá tabuľka ale nemá žiaden taký stĺpec. Aby sme jednoznačne určili riadok v druhej tabuľke, potrebujeme poznať číslo objednávky aj číslo tovaru. Žiadna z týchto hodnôt sama o sebe nestačí. Primárny kľúč druhej tabuľky teda pozostáva z dvoch zložiek. Takýto primárny kľúč sa nazýva **zreťazený primárny kľúč[7]**. Tabuľky, ktoré majú takýto zreťazený kľúč väčšinou pojednávajú o nejakom vzťahu medzi dvoma skupinami objektov (naša druhá tabuľka hovorí o vzťahu medzi objednávkami a tovarmi). A druhé užitočné pravidlo hovorí, že **ak máte tabuľku so zreťazeným primárnym kľúčom, nedržte v nej veci, ktoré závisia len od jeho časti**. V našej tabuľke sú informácie o tovare, ktoré závisia iba od identifikačného čísla tovaru. A keďže sa toto číslo občas zopakuje, zbytočne sa s ním zopakujú aj všetky ďalšie informácie o tovare. Tabuľku teda rozdelíme na dve a našu databázu tak dostaneme do **druhej normálnej formy**:

| Č.obj. | Dátum | ID Zák. | Meno | Adresa | Cena/Obj. |
|---|---|---|---|---|---|
| 17 | 01.04.2005 | 341 | Dežo Točka | Kubická 3 | 235,00 Sk |
| 18 | 02.04.2005 | 216 | Heda Divá | Kolmá 90 | 193,00 Sk |

| Č.obj. | ID Tovaru | Počet | Cena |
|---|---|---|---|
| 17 | 17 | 2 | 46,00 Sk |
| 17 | 21 | 1 | 65,00 Sk |
| 17 | 4 | 4 | 60,00 Sk |
| 17 | 8 | 2 | 64,00 Sk |
| 18 | 4 | 2 | 30,00 Sk |
| 18 | 21 | 1 | 65,00 Sk |
| 18 | 9 | 2 | 98,00 Sk |

| ID Tovaru | Popis | Cena/ks |
|---|---|---|
| 4 | Čierne bavl. | 15,00 Sk |
| 8 | Ružové | 32,00 Sk |
| 9 | Krajkové | 49,00 Sk |
| 17 | Zelené vlnené | 23,00 Sk |
| 21 | Zamatové | 65,00 Sk |

Poďme si teraz podrobnejšie všimnúť prvú z našich tabuliek. Tá má primárny kľúč, ktorý nie je zreťazený, takže z hľadiska druhej normálnej formy je všetko v poriadku. Ale ak si ju pozriete podrobnejšie, zistíte, že meno a adresa zákazníka nezávisia ani tak od primárneho kľúča – čísla objednávky, ako skôr od identifikačného čísla zákazníka. Čo má ten dôsledok, že v prípade pravidelných zákazníkov sa nám opäť budú údaje zbytočne hromadiť. Preto **je lepšie údaje, ktoré závisia od niečoho iného, ako od primárneho kľúča hodiť do samostatnej tabuľky**. (Má to okrem iného aj tú výhodu, že ak sa náhodou pravidelná zákazníčka Heda Divá vydá za pravidelného zákazníka Deža Točku, nebude treba meniť údaje o jej priezvisku a bydlisku na mnohých miestach, bude stačiť na jednom. Po tejto úprave dostaneme naše tabuľky do **tretej normálnej formy**:

6 Ako v našom prípade k jednej objednávke viaceré druhy ponožiek.

7 Zreťazený primárny kľúč nemusí pozostávať len z dvoch položiek. Môže aj z viacerých.

| Č.obj. | Dátum | ID Zák. | Cena/Obj. |
|---|---|---|---|
| 17 | 01.04.2005 | 341 | 235,00 Sk |
| 18 | 02.04.2005 | 216 | 193,00 Sk |

| ID Zák. | Meno | Adresa |
|---|---|---|
| 216 | Heda Divá | Kolmá 90 |
| 341 | Dežo Točka | Kubická 3 |

| ID Tovaru | Popis | Cena/ks |
|---|---|---|
| 4 | Čierne bavl. | 15,00 Sk |
| 8 | Ružové | 32,00 Sk |
| 9 | Krajkové | 49,00 Sk |
| 17 | Zelené vlnené | 23,00 Sk |
| 21 | Zamatové | 65,00 Sk |

| Č.obj. | ID Tovaru | Počet | Cena |
|---|---|---|---|
| 17 | 17 | 2 | 46,00 Sk |
| 17 | 21 | 1 | 65,00 Sk |
| 17 | 4 | 4 | 60,00 Sk |
| 17 | 8 | 2 | 64,00 Sk |
| 18 | 4 | 2 | 30,00 Sk |
| 18 | 21 | 1 | 65,00 Sk |
| 18 | 9 | 2 | 98,00 Sk |

Posledné pravidlo, ktoré tu spomenieme a ktoré je vhodné dodržať je, že údaje, ktoré je možné vypočítať z iných je lepšie v tabuľkách neskladovať. Databázové servery majú prostriedky, ako ich znovu vypočítať relatívne rýchlo. (Budeme o nich hovoriť v nasledujúcich lekciách.) Takže v našom prípade je vhodné vynechať zo záznamov o tom, koľko akého tovaru sa v danej objednávke nachádza cenu (môžeme ju kedykoľvek vyrátať ako počet krát cena za kus) a z tabuľky objednávok celkovú cenu objednávky (tam je výpočet trošku zložitejší, ale tiež nepredstavuje problém). Takže definitívna varianta databázy pre Fera Fusakľu by pozostávala zo štyroch tabuliek a mohla by vyzerať približne takto:

| Č.obj. | Dátum | ID Zák. |
|---|---|---|
| 17 | 01.04.2005 | 341 |
| 18 | 02.04.2005 | 216 |

| ID Zák. | Meno | Adresa |
|---|---|---|
| 216 | Heda Divá | Kolmá 90 |
| 341 | Dežo Točka | Kubická 3 |

| ID Tovaru | Popis | Cena/ks |
|---|---|---|
| 4 | Čierne bavl. | 15,00 Sk |
| 8 | Ružové | 32,00 Sk |
| 9 | Krajkové | 49,00 Sk |
| 17 | Zelené vlnené | 23,00 Sk |
| 21 | Zamatové | 65,00 Sk |

| Č.obj. | ID Tovaru | Počet |
|---|---|---|
| 17 | 17 | 2 |
| 17 | 21 | 1 |
| 17 | 4 | 4 |
| 17 | 8 | 2 |
| 18 | 4 | 2 |
| 18 | 21 | 1 |
| 18 | 9 | 2 |

3. lekcia

# Príkaz CREATE TABLE alebo „Urobme si databázu“

V minulej lekcii sme sa dozvedeli, ako by asi mala vyzerať dobre navrhnutá databáza. Že by mala spĺňať nejaké kritériá, ktoré sa nazývajú normálne formy. V tejto lekcii budeme hovoriť o tom, ako v SQL jazyku povedať serveru, ako vytvoriť takú tabuľku, akú chceme.

Začneme ukážkou. Príkaz na vytvorenie tabuľky `klienti` s ktorou ste sa stretli v prvej lekcii vyzerá takto:

```
mysql> CREATE TABLE klienti
    -> ( klient_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->   meno CHAR(15) NOT NULL,
    ->   priezvisko CHAR(20) NOT NULL,
    ->   bydlisko_ulica CHAR(30) NOT NULL,
    ->   bydlisko_obec CHAR(30) NOT NULL,
    ->   bydlisko_psc CHAR(6) NOT NULL,
    ->   email CHAR(25),
    ->   telefon CHAR(20),
    ->   aktivny ENUM('Y','N') NOT NULL DEFAULT 'Y'
    -> );
```

Ako vidno, tabuľka sa vytvorí príkazom `CREATE TABLE` (schválne – keby ste to nevedeli, uhádli by ste?) po ktorom nasleduje meno tabuľky. A potom sa do zátvoriek uvedú jednotlivé kolónky tabuľky, ktoré sa oddeľujú čiarkami. Za poslednou položkou čiarka byť nesmie.

Každá položka začína názvom kolónky. Po nej musí nasledovať dátový typ. A až po ňom idú všetky ostatné doplnkové informácie týkajúce sa danej kolónky (teda `klient_id INT UNSIGNED` je správne, ale `klient_id UNSIGNED INT` nie). V našej tabuľke sme použili tri rôzne dátové typy. Typ `INT`, niekoľko položiek rôznej dĺžky typu `CHAR` a typ `ENUM`. Do `INT`ov sa ukladajú celé čísla, do `CHAR`ov textové reťazce a do položky `ENUM` uložíte iba to, čo je vymenované za ňou v zátvorke.

Najviac doplnkových informácii je pri položke `klient_id`. Slovíčko `UNSIGNED` znamená, že sa tam nebudú ukladať záporné čísla. Slovné spojenie `NOT NULL` zas dá vedieť, že položka nemôže nadobudnúť zvláštnu hodnotu `NULL`, podľa ktorej sa dá zistiť, že položka nebola definovaná. (To ale neznamená, že napr. v prípade premennej typu `CHAR` nemôže byť hodnotou prázdny reťazec.) Slovo `AUTO_INCREMENT` zas dá systému vedieť, že čísla tejto položky nie je nutné zadávať. Server pre potreby tejto tabuľky vyrobí jej vlastné počítadlo, ktorému pri pridaní novej položky automaticky zväčší hodnotu, takže ju nemusí zadávať používateľ (alebo miesto nej môže zadať `NULL`). A nakoniec spojenie `PRIMARY KEY` dá serveru vedieť, že táto hodnota je primárnym kľúčom tabuľky. Ona určuje, či sú dva záznamy zhodné a podľa nej sa v tabuľke bude najčastejšie vyhľadávať.

Ďalšia zaujímavá vec sa nachádza pri položke `aktivny`. Táto položka tabuľky má v popise kľúčové slovo `DEFAULT` po ktorom nasleduje hodnota `'Y'`. Znamená to, že ak nepovieme, aká hodnota má v tejto položke byť, nastaví sa automaticky `'Y'`.

**Úloha č.1:** Vo svojej vlastnej databáze si vytvorte tabuľku `klienti`.

V nasledujúcich tabuľkách nájdete zoznam rôznych dátových typov, ktoré môžete v databáze MySQL používať:

**Tabuľka 1:** Celočíselné dátové typy

| *Typ* | *Rozsah* | *Obsadí bajtov* |
|---|---|---|
| `TINYINT` | -128 … 127<br>0 … 255 | 1 |
| `SMALLINT` | -32768 … 32767<br>0 … 65535 | 2 |
| `MEDIUMINT` | $-2^{23}$ … $2^{23}-1$<br>0 … $2^{24}-1$ | 3 |
| `INT`<br>`INTEGER` | $-2^{31}$ … $2^{31}-1$<br>0 … $2^{32}-1$ | 4 |
| `BIGINT` | $-2^{63}$ … $2^{63}-1$<br>0 … $2^{64}-1$ | 8 |

**Tabuľka 2:** Dátové typy pre čísla s pohyblivou rádovou čiarkou

| *Typ* | *Rozsah* | *Obsadí bajtov* |
|---|---|---|
| `FLOAT(presnosť)` | záleží na presnosti | rôzne |
| `FLOAT` | $\pm 1.175494351 \times 10^{-38}$<br>$\pm 3.402823466 \times 10^{38}$ | 4 |
| `DOUBLE`<br>`DOUBLE PRECISION`<br>`REAL` | $\pm 1.7976931348623157 \times 10^{308}$<br>$\pm 2.2250738585072014 \times 10^{-308}$ | 8 |
| `DECIMAL(M)`<br>`NUMERIC(M)`<br>`DEC(M)` | záleží na presnosti | M+2 |

Pri type `FLOAT` je možné určiť presnosť. Ak napríklad napíšeme `FLOAT(6,2)`, znamená to, že číselný údaj bude mať dĺžku 6 znakov a na úsek za desatinnou čiarkou sú určené dve cifry. Typ `DECIMAL` sa zas vyznačuje tým, že desatinné čísla ukladá ako text.

**Tabuľka 3:** Textové dátové typy

| *Typ* | *Rozsah* |
|---|---|
| CHAR(M) | M znakov (0 až 255) |
| CHAR | 1 |
| VARCHAR | Premenná dĺžka reťazca |
| TINYTEXT | do 255 znakov |
| TEXT | do 65535 znakov |
| MEDIUMTEXT | do 16 777 215 znakov |
| LONGTEXT | do 4 294 967 295 znakov |

Typy končiace na „TEXT“ majú premenlivú dĺžku a môžu byť pomerne rozsiahle a preto je prístup k nim pomalší. Používajte ich opatrne. Slúžia na ukladanie rozsiahlejších textov. Na ukladanie rozsiahlejších binárnych objektov (napríklad obrázkov alebo zvukov) slúžia dátové typy BLOB (TINYBLOB, BLOB, MEDIUMBLOB a LONGBLOB), ktoré majú rovnakú maximálnu veľkosť, ako ich proťajšky typu TEXT.

**Tabuľka 4:** Dátové typy pre dátum a čas

| *Typ* | *Rozsah* |
|---|---|
| DATE | 1000-01-01 až 9999-12-31 |
| TIME | -838:59:59 až 838:59:59 |
| DATETIME | kombinácia dátumu a času |
| TIMESTAMP | Pole do ktorého sa pri príkaze INSERT vloží aktuálny dátum a čas. |

Okrem uvedených dátových typov sú ešte dva špeciálne. Typ ENUM môže nadobúdať niektorú z uvedených možností. (Použitie ste videli v tabuľke klienti.) Typ SET môže nadobúdať viaceré z uvedených možností. Ak je napríklad položka typu SET('1','2','3'), môže nadobúdať napríklad hodnoty 1 alebo 2,3 alebo 1,3 alebo 1,2,3.

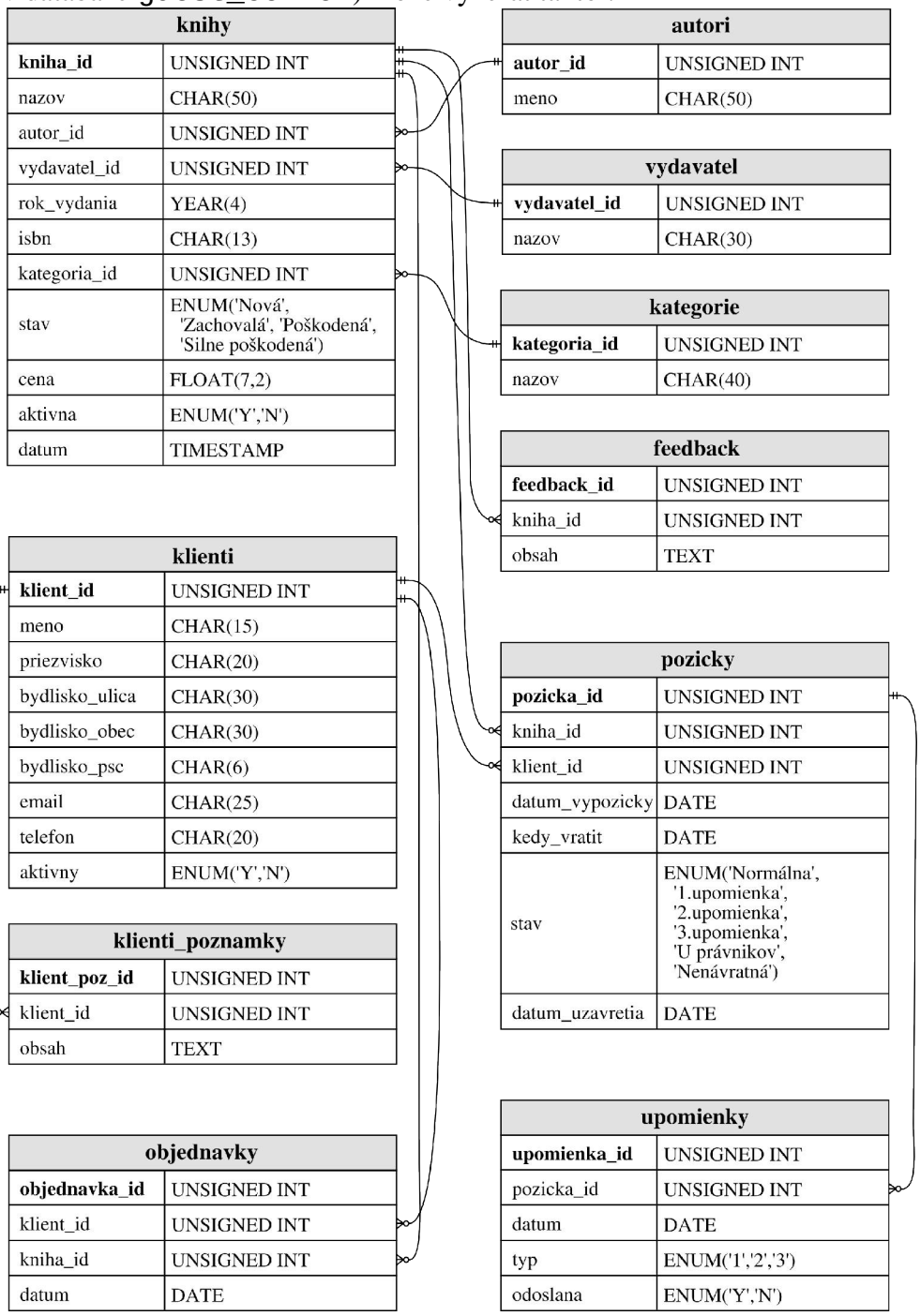
**Úloha č.2:** Teraz už viete, čo všetko je možné do databázy napchať. Pokúste sa teraz nejakú urobiť. Pôjde o databázu, ktorá sa môže použiť na prevádzku malej knižnice. Ide o to, aby tam bola tabuľka klientov, tabuľka poznámok ku klientom, tabuľka kníh (a s ňou súvisiace tabuľky autorov a vydavateľstiev, kategórií kníh, prípadne stručných obsahov a recenzií) ktorá bude obsahovať všetky dôležité údaje o knihe, tabuľka objednávok a tabuľka pôžičiek. Vymyslite, aké položky majú tabuľky mať, aby to fungovalo a aby spĺňali normálne formy z 2. lekcie. Ak v nejakej tabuľke niečo nie je tak, ako ste si predstavovali, môžete ju zmazať príkazom DROP TABLE:

```
DROP TABLE nepodarok;
```

4. lekcia

# Manipulácia s dátami alebo „Veci sa menia“

V predošlej lekcii sme sa na záver potýkali s úlohou zostaviť zmysluplnú sadu tabuliek, ktoré by boli použiteľné ako databáza pre menšiu knižnicu. Jedno z riešení (nájdete ho zrealizované v databáze `guest_common`) môže vyzerať takto[8]:




8 Táto databáza je skutočne malá. Mapa databázy jednej poisťovne v istej programátorskej firme zaberá celú stenu.

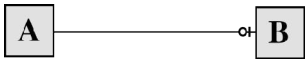
V každej tabuľke je hrubo zvýraznená položka, ktorá slúži ako primárny kľúč. Je vhodné všade ho definovať ako `NOT NULL AUTO_INCREMENT` aby bolo zabezpečené, že bude nastavený a že sa bude automaticky zvyšovať.

Čiarami je naznačené, ako sa jednotlivé tabuľky odvolávajú na položky v iných tabuľkách a ako spolu súvisia. To je výhodné, pretože položka, na ktorú sa odkazujete sa vyhľadá rýchlo. Ak sa odvolávate na položku v inej tabuľke, ktorá nie je primárnym kľúčom, je dobré do tej tabuľky doplniť ďalší kľúč – tabuľka môže mať kľúčov viacero a prácu s databázou to urýchli. V našom prípade sa vždy odvolávajú na primárny kľúč inej tabuľky.

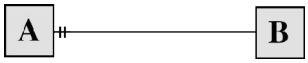
Všimnite si, že ak sa položka tabuľky odkazuje na položku v inej tabuľke, obe majú rovnaké meno (napríklad `klient_id`). Je to konvencia, ktorej dodržanie vám ušetrí množstvo zmätkov.

Ďalej je rozumné, aby položky, ktoré nemajú ostať nedefinované mali atribút `NOT NULL`. Na druhú stranu napríklad pri položke `datum_uzatvorenia` z tabuľky `pozicky` je vyložene vhodné, aby hodnotu `NULL` nadobúdať mohla. Bude to symbolizovať, že pôžička ešte nebola uzatvorená a knižka je stále požičaná.

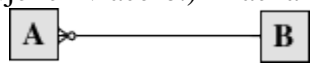
Značkami na konci čiar sa v mape databázy vyjadruje, v akom presne vzťahu dané dve tabuľky sú. Značka



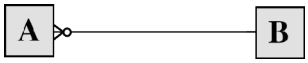

znamená, že každá inštancia (každý riadok) z tabuľky A sa vzťahuje k žiadnemu alebo jedinému záznamu v tabuľke B. Značka




znamená, že každá inštancia z tabuľky B sa vzťahuje k práve jednej inštancii z tabuľky A. (Teda, že aspoň jedna taká existuje, ale neexistuje ich viacero.) Značka

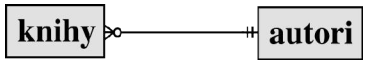



znamená, že každá inštancia z tabuľky A sa vzťahuje k minimálne jednej a maximálne mnohým inštanciám z tabuľky B (To znamená, že ich tam môže byť veľa, ale zaručene tam bude aspoň jedna.) A značka




znamená, že každá inštancia z tabuľky B sa vzťahuje k minimálne žiadnemu a maximálne mnohým inštanciám z tabuľky A. Skrátka, mapovači databáz vystačia s jednoduchou aritmetikou, ktorá obsahuje iba čísla 0, 1 a veľa.

Ako to vyzerá v našom konkrétnom prípade? Napríklad väzba medzi tabuľkami **knihy** a **autori** vyzerá nasledovne:




Znamená to, že ku každej knihe existuje práve jedna položka v tabuľke **autori**, ktorá jej zodpovedá (ak má teda kniha viacero autorov, treba v tabuľke autorov vytvoriť položku, v ktorej budú všetci).[9]

9 Keby sa takýto návrh databázy ukázal ako nepraktický, je možné ho zmeniť – napríklad každého zo spoluautorov uložiť do tabuľky autorov samostatne a zriadiť novú tabuľku, ktorá bude hovoriť, ktorí autori ku ktorej knihe patria. Keby napríklad kniha s **id** 219 mala autorov s **id** 31, 57 a 118, tabuľka by obsahovala tri riadky s dvojicami (219,31), (219,57) a (219,118). Tým by sa vzťah medzi tabuľkami **knihy** a **autori** zmenil z N:1 na N:N. Každému autorovi by mohlo zodpovedať viacero kníh a každej knihe viacero autorov.

A ku každému autorovi v databáze môže ale nemusí existovať kniha, ktorej je autorom, a kníh, ktoré napísal, môže byť viacero.

Väčšina databáz obsahuje špeciálne tabuľky, ktorých účel je uchovávať systémové nastavenia a premenné. Takáto tabuľka obsahuje iba jeden riadok v ktorom sa nachádzajú konkrétne hodnoty systémových premenných. Nemusí preto obsahovať žiaden kľúč – v tabuľke sa žiadne vyhľadávanie nedeje. V našom prípade môžeme v takejto tabuľke uchovať napríklad čas, po akom sa posielajú jednotlivé upomienky, výšky pokút alebo dobu rezervácie knihy. V prípade potreby je možné tieto nastavenia jednoducho zmeniť a systém bude brať do úvahy nové hodnoty. Naša systémová tabuľka môže teda vyzerať napríklad takto:

| system | | |
|---|---|---|
| 1_upomienka_limit | UNSIGNED INT | 31 |
| 2_upomienka_limit | UNSIGNED INT | 62 |
| 3_upomienka_limit | UNSIGNED INT | 93 |
| 1_upomienka_suma | FLOAT(6,2) | 10.00 |
| 2_upomienka_suma | FLOAT(6,2) | 30.00 |
| 3_upomienka_suma | FLOAT(6,2) | 60.00 |
| doba_rezervacie | UNSIGNED INT | 2 |

V poslednom stĺpci sú uvedené aj aktuálne nastavené hodnoty.

Doteraz sme sa naučili, ako tabuľku vytvoriť, ako ju zmazať a ako do nej vkladať údaje. V tejto lekcii sa naučíme väčšinu ostatných dôležitých príkazov a nabudúce si necháme príkaz najmocnejší a najprešpekulovanejší – príkaz `SELECT`.

Takže najprv zmeny. Napríklad v tabuľke `klienti` chceme zmeniť klientovi, ktorý má `klient_id` rovné 1 (Jolane Laznibatovej) položku `bydlisko_ulica` z hodnoty `Teplicka 7` na hodnotu `Skalicka 1`. Slúži na to príkaz `UPDATE`. Spraví sa to takto:

```
UPDATE klienti
        SET bydlisko_ulica = “Skalicka 1“
        WHERE klient_id = 1;
```

Pokiaľ máme v tabuľke iba jednu Jolanu, bude fungovať aj príkaz

```
UPDATE klienti
        SET bydlisko_ulica = “Skalicka 1“
        WHERE meno = “Jolana“;
```

Pokiaľ ale v tabuľke klienti nezriadime kľúč na vyhľadávanie podľa krstného mena, príkaz sa bude vykonávať pomalšie, pretože sa jedna po druhej skontrolujú všetky položky. Ak máme v tabuľke Jolán viacero, nastaví sa `bydlisko_ulica` na `Skalicka 1` v druhom prípade každej z nich. Podobne keby sme sa rozhodli, že veľkoryso každému dovolíme, aby vrátil knižku o 1 deň neskôr, môžeme použiť príkaz

```
UPDATE pozicky
   SET kedy_vratit = ADDDATE(kedy_vratit, INTERVAL 1 DAY);
```

Použili sme funkciu ADDDATE, ktorá k nejakému dátumu vie pripočítať určenú hodnotu. Keďže chceme operáciu previesť na všetkých položkách tabuľky, príkaz WHERE vôbec nepoužijeme.

Na mazanie slúži príkaz DELETE FROM. Keby sme napríklad chceli vymazať klienta č. 1 z tabuľky, použili by sme príkaz

```
DELETE FROM klienti WHERE klient_id = 1;
```

Časť WHERE je tentokrát veľmi dôležitá. Ak by ste ju neuviedli, vymažete celý obsah tabuľky!!!

Okrem zmeny dát v tabuľke môžete meniť aj samotné tabuľky. Slúži na to univerzálny príkaz ALTER TABLE, ktorému treba povedať, ktorá tabuľka sa bude meniť a čo sa v nej bude meniť. Keby sme napríklad do tabuľky klientov chceli pridať za položku bydlisko_psc novú položku, spravili by sme to takto:

```
ALTER TABLE klienti
        ADD idiot ENUM('Y','N') NOT NULL DEFAULT 'N'
        AFTER bydlisko_psc;
```

Keby sme namiesto AFTER bydlisko_psc napísali FIRST, položka sa pridá na začiatok tabuľky. Ak nenapíšeme nič, pridá sa na koniec.

Keby sme položku zas chceli vymazať, príkaz by vyzeral takto:

```
ALTER TABLE klienti DROP idiot;
```

Ak chceme niektorý stĺpec zmeniť, použijeme (prekvapivo…) slovo CHANGE. Takže ak by sme chceli zbaviť kolónku klient_id vlastnosti AUTO_INCREMENT, zmenili by sme ju takto:

```
ALTER TABLE klienti
        CHANGE klient_id klient_id INT UNSIGNED NOT NULL;
```

Všimnite si, že po CHANGE klient_id nasleduje kompletný popis novej kolónky (včetne názvu kolónky, ktorý sa tak môže zmeniť). Ak je možné dáta v danom stĺpci tabuľky skonvertovať na nový formát, udeje sa to.

Keby sme chceli tabuľke odobrať primárny kľúč, použijeme príkaz

```
ALTER TABLE klienti DROP PRIMARY KEY;
```

Tento príkaz neodstráni kolónku klient_id, kolónka iba prestane byť primárnym kľúčom[10]. To znamená, že by v tabuľke mohli existovať dvaja klienti s rovnakou hodnotou klient_id. Ak tabuľke chceme primárny kľúč pridať, spravíme to takto:

```
ALTER TABLE klienti ADD PRIMARY KEY (klient_id);
```

Keby sme v tabuľke často vyhľadávali nielen podľa identifikačného čísla klienta, ale aj podľa jeho mena a priezviska, a nechceme, aby databázový stroj zakaždým prehľadával celú tabuľku a vedel veci urýchliť, môžeme tabuľke vytvoriť index. To môže niekedy veci značne urýchliť. (Napríklad by

10 Pozor! Kolónka typu AUTO_INCREMENT musí byť kľúčom. Ak nechcete, aby kľúčom bola, musíte ju najprv zmeniť tak, aby nebola ani AUTO_INCREMENT.

sme sa mohli zbaviť toho, aby sa prehľadávala celá tabuľka, keď sa mení bydlisko všetkým Jolanám.) Nový index sa pridáva takto:

```
ALTER TABLE klienti ADD INDEX podlaMena (priezvisko,meno);
```

Novovytvorený index sa bude volať `podlaMena` a bude si pamätať zoradenie tabuľky podľa priezvisk a v prípade rovnakého priezviska podľa krstných mien.

Index môžeme vytvoriť aj tak, aby mal ešte jednu vlastnosť, ktorá sa môže hodiť – môže zabezpečiť, aby sa v tabuľke neopakovali dve z hľadiska indexu rovnaké hodnoty. Ak by sme chceli vytvoriť takýto index, použijeme namiesto predošlého príkazu tento:

```
ALTER TABLE klienti ADD UNIQUE podlaMena (priezvisko,meno);
```

Takto upravená tabuľka si bude kontrolovať, či sa v nej nevyskytujú menovci. Ak by ste tam chceli menovca zaradiť, odmietne. (Vyskúšajte si to!!!)

Ak chcete index zrušiť, použite príkaz

```
ALTER TABLE klienti DROP INDEX podlaMena;
```

Dajte si pozor, aby ste v tabuľke neudržiavali priveľa indexov. Zrýchli sa nimi síce vyhľadávanie, ale spomalí sa manipulácia s dátami a indexy zaberajú aj istý priestor na disku.

Ak chcete zmeniť štandardnú hodnotu niektorého stĺpca, spraví sa to príkazom

```
ALTER TABLE klienti
        ALTER bydlisko_obec SET DEFAULT "Bratislava";
```

Ak chcete iba štandardnú hodnotu zrušiť, namiesto `SET DEFAULT` `voľačo` napíšete iba `DROP DEFAULT`.

A na záver, ak chcete zmeniť názov tabuľky, spravíte to takto:

```
ALTER TABLE klienti RENAME zakaznici;
```

**Úloha č. 1:** Vyskúšajte každý z uvedených príkazov (samozrejme na vašich vlastných tabuľkách). Môžete sa napríklad pokúsiť zmeniť tabuľky z vašej úlohy z predošlej lekcie s pomocou tu uvedených príkazov na také, aké sú popísané na začiatku tejto lekcie.

**Úloha č.2:** Do databázy `guest_common` pridajte do každej tabuľky zodva zmysluplné údaje, aby bolo na čom skúšať veľkolepý príkaz `SELECT`.

5. lekcia

# Príkaz SELECT
# alebo „Ako sa spýtať databázového orákula“

Doteraz boli všetky lekcie v podstate o tom, ako do databázy nejaké údaje napchať. Táto lekcia bude o tom, ako z databázy nejaké údaje dostať. Na akékoľvek získavanie údajov slúži jediný príkaz – SELECT.

SELECT slúži na zobrazenie čohokoľvek. Môžete napríklad napísať

```
SELECT 1 + 1;
```

alebo

```
SELECT 2 * PI() * 5;
```

a v oboch prípadoch dostanete správnu odpoveď.[11] Tieto príklady sa ale veľmi nevzťahujú k tabuľkám, ktoré naša databáza spravuje. S tabuľkoidnou formou príkazu SELECT sme sa už ale tiež stretli. Keď sme chceli vypísať, čo daná tabuľka obsahuje, napísali sme

```
SELECT * FROM klienti;
```

Hviezdička má v tomto príkaze význam „úplne všetko“. Nie vždy má ale zmysel úplne všetko vypisovať. Keby sme napríklad chceli iba menný zoznam klientov knižnice usporiadaný podľa abecedy, použili by sme príkaz

```
SELECT meno, priezvisko FROM klienti ORDER BY priezvisko, meno;
```

Namiesto všetkých kolóniek sa teraz budú vypisovať iba dve. Príkazom `ORDER BY` dávame serveru najavo, že výsledky chceme mať usporiadané podľa priezvisk a v prípade rovností priezvisk podľa mien. V prípade, že by sme takýto príkaz chceli vykonávať často nad veľkou tabuľkou, je vhodné spraviť si index.

V niektorých situáciach by sa vám mohlo hodiť, aby výsledok funkcie SELECT nebol v dvoch stĺpcoch, ale v jednom. Nič nám nebráni použiť funkciu `CONCAT`, ktorá spája reťazce, ktoré jej zadáte do jednoho. Príkaz môže vyzerať napríklad takto:

```
SELECT CONCAT(meno,“ “,prezvisko) FROM klienti
        ORDER BY priezvisko, meno;
```

Keby ste chceli, aby databázový stroj nevrátil všetky výsledky (napríklad preto, lebo chcete zobraziť výsledky na webovej stránke a všetky sa na stránku nezmestia), môžete to zariadiť príkazom `LIMIT`. Napríklad príkaz

```
SELECT nazov FROM knihy ORDER BY nazov LIMIT 40, 10;
```

11 Pravdepodobne ste si domysleli, že `PI()` je funkcia, ktorá vráti hodnotu π. Funkcií pozná MySQL mnoho, ak o nich chcete vedieť viac, zadajte príkaz `HELP Functions;`

Vypíše z tabuľky kníh 10, pričom začne štyridsiatou. Výsledky funkcie SELECT sú číslované ako v C-čku, to znamená, že na začiatku je nultý výsledok.

Keby sme chceli z tabuľky klientov vypísať všetkých bratislavčanov, spravili by sme to príkazom

```
SELECT meno,priezvisko FROM klienti
        WHERE bydlisko_obec="Bratislava";
```

a ak chceme všetky knihy, ktoré majú v názve SQL, spravíme to takto:

```
SELECT knihy.nazov FROM knihy
        WHERE knihy.nazov LIKE "%SQL%";
```

A konečne sa dostávame k tomu, prečo sa databázy ovládané SQL jazykom nazývajú relačné databázy. Všetky príkazy SELECT sa doteraz týkali jednej jedinej tabuľky. Lenže medzi tabuľkami môžu byť zložité vzťahy a my by sme chceli, aby sa v tom databáza nejako vyznala. Príkaz SELECT je ale schopný vyhľadávať vo viacerých tabuľkách naraz.

Začneme niečim jednoduchým. Napríklad chceme zoznam názvov knižiek aj s autormi. Názvy knižiek sú v inej tabuľke, než mená autorov. SELECT bude teda musieť čítať obe tabuľky a priradiť správneho autora k správnej knižke. Spraví sa to takto:

```
SELECT autori.meno, knihy.nazov FROM knihy, autori
        WHERE knihy.autor_id = autori.autor_id;
```

Ako to vlastne funguje? Skúste zadať databáze ten istý príkaz, ale bez časti WHERE. Zistíte, že vám vypíše každého autora s každou knižkou. No a časť WHERE spôsobí, že sa z tejto masy vypíšu iba tie veci, kde je číslo autora z tabuľky kníh rovnaké ako číslo autora z tabuľky autorov.

Teraz niečo kúsok zložitejšie. Potrebujeme zoznam všetkých hororov, ktoré máme v knižnici. Spraví sa to takto:

```
SELECT knihy.nazov FROM knihy,kategorie
        WHERE knihy.kategoria_id = kategorie.kategoria_id
          AND kategorie.nazov = "Horor";
```

Keby sme poznali číslo kategórie „horor“, vystačili by sme aj so selectom, ktorý by pristupoval iba k tabuľke knihy. To však bežne nevieme, lebo to býva vnútornou záležitosťou databázy.

Ďalšia úloha – potrebujeme zistiť mená ľudí, ktorí majú nejaké požičané knihy a to, aké knihy majú požičané. Príkaz bude vyzerať takto:

```
SELECT klienti.meno, klienti.priezvisko, knihy.nazov
        FROM knihy,klienti,pozicky
        WHERE pozicky.klient_id = klienti.klient_id
          AND pozicky.kniha_id=knihy.kniha_id
          AND pozicky.datum_uzatvorenia IS NULL;
```

Všimnite si, že aj keď tabuľka pozicky nesie v tomto prípade podstatnú informáciu, žiadne dáta priamo z nej sa nevypisujú – za príkazom SELECT nasledujú len kolónky z tabuliek klienti a

`knihy`. Tabuľka `pozicky` sa uplatní hlavne v časti `WHERE`.

Takže nejaké úlohy:

**Úloha č.1:** Vyskúšajte všetky uvedené príkazy.

**Úloha č.2:** Napíšte príkaz, ktorý nájde všetky knihy, ktorých autorom je Terry Pratchet.

**Úloha č.3:** Napíšte príkaz, ktorý zistí, ktorí ľudia majú objednanú knihu s `kniha_id` 1. Usporiadajte ich podľa času objednávky. Upravte príkaz tak, aby vypísal iba toho, kto si knihu objednal ako prvý.

**Úloha č.4:** Napíšte príkaz, ktorý vypíše, aké knihy má práve požičané klient s `klient_id` 3.

**Úloha č.5:** Napíšte príkaz, ktorý vypíše všetkých klientov, ktorí si niekedy požičali knižku, v ktorej názve sa vyskytuje slovo „prsteň“.

**Úloha č.6:** Ak klient nevrátil knihu v určenom termíne a za posledných 31 dní mu nebola poslaná upomienka, treba mu poslať ďalšiu. Napíšte príkaz, ktorí zistí, ktorým klientom treba poslať upomienky. (V tejto úlohe budete potrebovať funkcie na prácu s dátumami. Ich zoznam získate príkazom `HELP Date and Time Functions;` a popis jednotlivých funkcií príkazom `HELP <meno_funkcie>`.)

Príkaz `SELECT` môže nielen vyberať riadky podľa našich kritérií. Môže riadky aj zoskupovať a pre danú skupinu údajov vykonať nejaký výpočet. Ak napríklad chceme zistiť počet kníh v knižnici, spravíme to príkazom

```
SELECT COUNT(*) FROM knihy;
```

V tomto prípade tvoria všetky knihy jednu veľkú skupinu. Keby sme ale napríklad chceli zistiť, koľko kníh je v jednotlivých kategóriách, musíme povedať, podľa čoho má skupiny vytvoriť. Príkaz môže vyzerať napríklad takto:

```
SELECT kategorie.nazov,COUNT(*) FROM knihy,kategorie
        WHERE knihy.kategoria_id = kategorie.kategoria_id
        GROUP BY knihy.kategoria_ID;
```

Knihy sa rozdelia na skupiny podľa kategórii a pre každú z nich sa vypíše názov kategórie a počet kníh v kategórii.

Okrem funkcie `COUNT` môžeme pre skupiny používať funkcie `AVG`, ktorá vyráta priemernú hodnotu, funkcie `MIN` a `MAX` ktoré zistia minimum a maximum, funkciu `STD`, ktorá vypočíta smerodajnú odchýlku (štatistici poznajú… ) a funkciu `SUM`, ktorá čísla v danej kolónke sčíta. Každej z nich ako parameter treba uviesť položku tabuľky, ktorej sa bude týkať. Takže ak

napríklad chceme zistiť celkovú hodnotu kníh v knižnici, spravíme to príkazom

```
SELECT SUM(cena) FROM knihy;
```

a ak chceme zistiť priemernú cenu knihy v jednotlivých kategóriách, použijeme príkaz

```
SELECT kategorie.nazov,AVG(knihy.cena)
    FROM knihy,kategorie
         WHERE knihy.kategoria_id = kategorie.kategoria_id
         GROUP BY knihy.kategoria_ID;
```

V príkaze sa `SELECT` môže vyskytovať aj viackrát. Keby sme napríklad hľadali najdrahšiu knižku v knižnici, použili by sme príkaz

```
SELECT knihy.nazov FROM knihy
   WHERE knihy.cena = (SELECT MAX(knihy.cena) FROM knihy);
```

**Úloha č.7:** Napíšte príkaz, ktorý nájde najlacnejší horor v knižnici.

**Úloha č.8:** Napíšte príkaz, ktorý zistí, koľko kníh mal počas celej histórie každý klient požičaných.

**Úloha č.9:** Upravte príkaz z úlohy č.6 tak, aby zistil, koľká upomienka sa danému človeku ide poslať.

6. lekcia

# Príkaz EXPLAIN
# alebo „Ako sa spýtať lepšie“

V predošlej lekcii sme sa naučili, ako sa databázy niečo spýtať. V prípade, že je databáza malá, veľmi na tom, ako presne sa spýtate, nezáleží. Ale ako je pri úspešných projektoch dobrým zvykom, databáza sa začne rozrastať. Časy, kedy knižnica obsahovala dvadsať knižiek je nenávratne preč, dnes obsahuje státisíce titulov od desaťtisícov autorov a vy zisťujete, že príkazy na vyhľadávanie v tabuľkách už nefungujú tak hladko a rýchlo, ako voľakedy.

Preto je dobré vedieť, ako databáza hľadá to, čo od nech chceme a podľa toho tabuľky či otázku optimalizovať. Ukážku takej optimalizácie uvidíte na jednoduchej úlohe – chceme spraviť vyhľadávanie podľa autora. Potrebujeme napríklad nájsť všetky knižky z našej knižnice, ktoré napísal pán J.R.R. Tolkien. Na začiatok jednoduché riešenie:

```
SELECT knihy.nazov FROM knihy,autori
        WHERE knihy.autor_id = autori.autor_id
          AND autori.meno = "J.R.R. Tolkien";
```

Riešenie je v poriadku, všetko sa vypíše tak ako má. Lenže čo sa deje v pozadí? Čo všetko musí databázový stroj preveriť, aby odpoveď našiel? Na to, aby sme to zistili, slúži príkaz `EXPLAIN`.[12] Používa sa tak, že ho napíšete pred príkaz `SELECT`, ktorý chcete testovať, čiže

```
EXPLAIN SELECT knihy.nazov FROM knihy,autori
        WHERE knihy.autor_id = autori.autor_id
          AND autori.meno = "J.R.R. Tolkien";
```

a ako výstup dostanete takúto tabuľku:

```
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | knihy  | ALL  | NULL          | NULL |    NULL | NULL |    3 |             |
|  1 | SIMPLE      | autori | ALL  | PRIMARY       | NULL |    NULL | NULL |    2 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
```

Každý riadok hovorí o jednej z tabuliek, ktoré sa nášho vyhľadávania zúčastňujú. Prvý stĺpec je identifikátor `SELECT`u, ktorý k tabuľke pristupuje. (Aby sa to rozlíšilo, ak je v príkaze `SELECT`ov viacero.) Druhý stĺpec je typ `SELECT`u. (Uvidíte, ako sa bude meniť, keď zmeníme otázku). Tretí je meno tabuľky. A teraz nastáva to dôležité – štvrtý stĺpec hovorí, ako sa k tabuľke pristupuje. A slovíčko `ALL` znamená najhoršiu možnú variantu. Tabuľka sa prezrie celá. A keby len to. Keď je `ALL` na oboch tabuľkách, znamená to, že sa s každou z možností z prvej tabuľky vyskúša každá možnosť z druhej tabuľky. Keď má prvá tabuľka 3 riadky a druhá 2 (ako v našom cvičnom príklade),

12 Činnosť príkazu `EXPLAIN` je pravdepodobne závislá na konkrétnej databáze. To, čo popisujeme v tejto lekcii sa týka špeciálne databázy MySQL. Vieme, že príkaz rovnakého mena obsahuje aj napr. PostgreSQL, ale na tejto databáze sme ho netestovali.

tak vyskúšať 3.2=6 možností nie je až taká tragédia. Ale keď má jedna tabuľka 100 000 záznamov a druhá 10 000 … V každom prípade je vhodné pozrieť sa na kolónku `rows` a uvedomiť si, že aby som zistil, koľko operácií sa vykoná, bude treba čísla prislúchajúce rovnakému `id`čku vynásobiť.

Takže poďme vylepšovať. Najprv by sme sa mohli zbaviť toho násobenia. Keď už prehľadávame celé tabuľky, nech každú stačí prezrieť raz. Spravíme to tak, že príkaz rozdelíme na dva `SELECT`y:

```
SELECT nazov FROM knihy
        WHERE autor_id =
        (SELECT autor_id FROM autori
                        WHERE meno = "J.R.R. Tolkien");
```

Keď si tento `SELECT` s pomocou príkazu `EXPLAIN` dáte vysvetliť, dostanete takúto tabuľku:

```
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
|  1 | PRIMARY     | knihy  | ALL  | NULL          | NULL |    NULL | NULL |    3 | Using where |
|  2 | SUBQUERY    | autori | ALL  | NULL          | NULL |    NULL | NULL |    2 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
```

Každý `SELECT` má teraz svoje vlastné `id`čko. Najprv sa vykoná podotázka, ktorá prezrie všetkých autorov, zistí, ktorý z nich sa volá J.R.R. Tolkien a vráti jeho `autor_id`. Potom sa s každým výsledkom (pravdepodobne ale bude len jeden) vykoná prvý `SELECT`. Toto z hľadiska databázy vyžaduje istú réžiu, takže to pri maličkých tabuľkách môže dokonca viesť k spomaleniu. Ale pri veľkých tabuľkách sa namiesto 100000.10000 záznamov (čo je miliarda!!!) prehľadá iba 100000+10000 (čo je 110000). Zlepšenie takmer desaťtisícnásobné.

Poďme ďalej zlepšovať. Z piatej kolónky vysvetľovacej tabuľky, ktorá hovorí, aké kľúče je možné v danej situácii použiť na nás pozerá smutné `NULL`. Každá z oboch tabuliek má totiž zatiaľ len jeden kľúč – primárny kľúč. Podľa neho vie vyhľadávať rýchlo, ale podľa ničoho iného vyhľadávať nevie. A nám by sa náramne hodilo, aby bola tabuľka kníh nejakým spôsobom utriedená aj podľa autorov. To ale nie je problém zariadiť. Zmeníme tabuľku a pridáme jej ďalší kľúč:

```
ALTER TABLE knihy ADD INDEX (autor_id);
```

Znovu si necháme vysvetliť posledný `SELECT` a dočkáme sa zmeny. Tabuľka bude vyzerať takto:

```
+----+-------------+--------+------+---------------+----------+---------+-------+------+-------------+
| id | select_type | table  | type | possible_keys | key      | key_len | ref   | rows | Extra       |
+----+-------------+--------+------+---------------+----------+---------+-------+------+-------------+
|  1 | PRIMARY     | knihy  | ref  | autor_id      | autor_id |       4 | const |    1 | Using where |
|  2 | SUBQUERY    | autori | ALL  | NULL          | NULL     |    NULL | NULL  |    2 | Using where |
+----+-------------+--------+------+---------------+----------+---------+-------+------+-------------+
```

Hľa, zmena. Konkrétne vyhľadávanie v tabuľke `knihy` už nie je typu `ALL`, ale typu `ref`. To znamená, že do tabuľku sa siahne a vyberú sa iba správne hodnoty a vyberú sa výrazne rýchlejšie, ako keby sa mala celá tabuľka prehľadať. Všimnite si počet kontrolovaných riadkov v tabuľke `knihy`. Aj v tabuľke s tromi položkami je to výrazné zlepšenie.

Aby sme to doviedli do dokonalosti, ostáva nám už len urýchliť vyhľadávanie podľa mena v tabuľke autorov. Zavedieme si tam index. A keďže nemá zmysel, aby sa v tabuľke nejaké meno opakovalo viackrát, bude ten index jednoznačný, čo situáciu ešte trochu zlepší:

```
ALTER TABLE autori ADD UNIQUE (meno);
```

Keď si znovu necháme vysvetliť náš `SELECT`, dozvieme sa nasledujúce veci:

```
+----+-------------+--------+-------+---------------+----------+---------+-------+------+-------------+
| id | select_type | table  | type  | possible_keys | key      | key_len | ref   | rows | Extra       |
+----+-------------+--------+-------+---------------+----------+---------+-------+------+-------------+
|  1 | PRIMARY     | knihy  | ref   | autor_id      | autor_id |       4 | const |    1 | Using where |
|  2 | SUBQUERY    | autori | const | meno          | meno     |      51 |       |    1 |             |
+----+-------------+--------+-------+---------------+----------+---------+-------+------+-------------+
```

V kolónke `type` sa teraz nachádza hodnota `const`, čo je to úplne najlepšie, čo môže byť. Hľadá sa totiž jediný riadok podľa jednoznačného kľúča, čo je úplne najlepší spôsob vyhľadávania, aký sa dá dosiahnuť. V tomto stave je vyhľadávanie podľa autora oveľa rýchlejšie, než bolo na začiatku.

**Úloha č.1:** Vyskúšajte všetko, čo bolo popísané v tejto lekcii.

**Úloha č.2:** Optimalizujte na rýchlosť vyhľadávanie človeka, ktorý má ako prvý objednanú danú knihu.

**Úloha č.3:** Prečítajte si ešte raz lekciu č.2. Keď ste videli, ako sa s databázami narába, mnohé veci vám budú jasnejšie.

7. lekcia

# Jazyk PHP
# alebo „Skoro C-čko“

V dávnych pradávnych časoch, keď bol internet ešte malý, sa stačilo naučiť niekoľko značiek typu `<h3>` alebo `<img src=“ja.jpg“>` a každý sa mohol pustiť do výroby vlastnej stránky. Túto vlastnosť si HTML jazyk do istej miery zachoval. Keď si človek chce zavesiť na web kúsok svojej literárnej tvorby, rozvrh alebo niekoľko fotiek, naozaj oveľa viac nepotrebuje.

Ale to by neboli programátori, keby nezačali niečo vymýšľať. Ľudia začali túžiť po tom, aby ich stránky neboli stále rovnaké. Aby sa menili podľa toho, ako si želajú používatelia, aby nakupujúcemu v internetovom obchode ukazovali, čo má práve v košíku, aby vedeli vyhľadať autobusové spojenie do Hornej Maríkovej, aby sa cez ne dali rezervovať letenky, aby ukazovali hráčovi, koľko planét s akými surovinami práve jeho vesmírna veľmoc obsadila a aké predmety práve úspešne absolvoval vo virtuálnej škole. Skrátka stránky prestali byť niečim pevným a nemenným, ale začali sa správať skôr ako používateľské rozhranie k nejakému programu, ktorý beží „za scénou“.

Riešilo sa to rôznym spôsobom. Jeden z nich je CGI (Common Gateway Interface). To funguje tak, že ak webserver dostane požiadavku na nejakú konkrétnu stránku, jej vytvorenie zverí nejakému inému programu, ktorý na tom počítači beží. Ten program môže byť napísaný napríklad v Cčku, PERLi, RUBY alebo hocičom inom zmysluplnom, môže spracovať vstup, ktorý mu webserver odovzdá a podľa neho vygenerovať, čo treba.

Ďalší spôsob, ako spraviť dynamický web je JavaScript (alebo jeho Microsoftia odroda JScript) prípadne novšia technológia Macromedia Flash.[13] Tam zase ide o to, že stránky obsahujú aj kód, ktorý nebeží na serveri, ale vie ho spustiť počítač, na ktorom beží prehliadač. Tento kód si môže od používateľa vypýtať vstup (či už do rôznych kolóniek, alebo od myši) a robiť niečo zmysluplné (väčšina z vás už iste hrala nejakú flashovú hru), ale o tom, čo sa na strane klienta deje, nemá server ani poňatia.

Oboma týmito prístupmi je inšpirovaný jazyk PHP. Skratka pôvodne znamenala „Personal Home Page tools“, dnes je jej oficiálnym významom rekurzívna skratka „PHP: Hypertext Preprocessor“. Od CGI zdedil to, že kód beží na strane servera. Je preto vhodné použiť ho, keď potrebujete aktivitu používateľa na strane servera zaznamenávať a spracovávať. Na JavaScript sa zase podobá v tom, že kód sa píše „dovnútra“ HTML.[14] O tom kóde sa ale klient nedozvie. Kód vykoná server a klientovi pošle iba to, čo z neho vylezie.

Apache aj PHP sú multiplatformné, takže vám pobežia skoro pod všetkým, na čo si zmyslíte (t.j. všetky možné UNIXy, Linux, Windows, OS X,… ) a keď sa k nim pridá MySQL vznikne šikovná, lacná a veľmi použiteľná zbierka nástrojov. (Ak to beží pod Linuxom, má to

13 Podobne fungujú aj niektoré aplety napísané v Jave.

14 Toto nemusí byť celkom vždy pravda. Programy v PHP je možné spúšťať aj mimo akéhokoľvek HTML. Dá sa použiť aj ako skriptovací jazyk (podobne ako shell alebo Perl). PHP sa dá použiť aj ako jazyk na písanie CGI skriptov.

skratku LAMP – Linux, Apache, MySQL, PHP, ak to beží pod Windowsom, hovorí sa tomu WAMP)

Takže dozrel čas na príklad. Spreneveríme sa všetkým klasickým učebniciam programovania a nebude to „Hello World!“, ale skriptík, ktorý vygeneruje tabuľku druhých mocnín. Na jeho spustenie potrebujete webový server (u nás v škole Apache), ktorý vie s jazykom PHP spolupracovať (v prípade Apache musí byť nainštalovaný modul mod_php). Stránky obsahujúce PHP kód majú štandardne koncovku `.php`, takže náš prvý pokus sa bude volať nejak ako `pokus.php`:

```
<html>
   <head>
      <title>Prvy pokus</title>
   </head>
   <body>
      <table border="2">
         <?php
            for($i = 1; $i <= 10; $i++)
               echo "<tr><td>".$i."</td><td>".$i*$i."</td></tr>\n"
         ?>
      </table>
   </body>
</html>
```

Na prvý pohľad to vyzerá ako klasické HTMLko. Vyrobíme nadpis, do tela stránky vložíme jednu tabuľku a tým to zhasne. Lenže vo vnútri tej tabuľky sa deje niečo nekalé. Je to uzavreté medzi značkami `<?php` a `?>` . To niečo je práve PHP kód. Kus uzavretý v uvedených značkách server vyberie z textu, spustí a tým, čo kód vypíše priestor medzi značkami nahradí.

Poďme sa teraz pozrieť na to, čo ten kód vlastne robí. Je to klasický `for` cyklus, aký poznáte z Cčka. Líši sa iba v dvoch detailoch. Prvý je ten, že premenná začína znakom `$`. Toto je v PHPčku povinné, podľa toho doláru sa rozlišuje, čo je a čo nie je premenná. Druhý detail je ten, že na rozdiel od Cčka sme premennú nikde nedeklarovali. V jazyku PHP to nie je potrebné. Do premenných tam môžete ukladať, čo sa vám práve hodí, len si musíte dávať pozor, aby ste si v tom udržali poriadok.

Cyklus obsahuje jediný príkaz. Keby ich mal obsahovať viac, bolo by ich treba (rovnako ako v Cčku) uzavtieť medzi `{` a `}`. Ten príkaz je príkaz `echo`, ktorý slúži na výpis textu. V našom prípade vypisuje jeden dlhý reťazec. Ten reťazec je pospájaný z reťazcov obsahujúcich HTML značky uzavreté v úvodzovkách a z hodnôt, ktoré počítame z premennej `$i` a ktoré do tabuľky chceme napchať. Na operáciu „zlep reťazce dokopy“ sa v PHPčku používa bodka. A tá bodka je dostatočne inteligentná, aby vedela zmeniť číselné hodnoty `$i` a `$i*$i` na reťazce a prilepiť ich ku zvyšku.

**Úloha č.1:** Vyskúšajte si to. Zobrazte si stránku a pozrite si zdrojový kód, ktorý dostal prehliadač (v Mozille sa zobrazuje cez `Ctrl-U`). V čom sa líši od toho, čo ste napísali do PHP súboru?

Takže tak. Ono je to v podstate Cčko až na niektoré detaily o ktorých budeme hovoriť v niektorej z ďalších lekcií. Skúste teraz so znalosťou Cčka, ktorú máte a s tým, čo ste sa dozvedeli v tejto lekcii vyriešiť nasledujúce úlohy:

**Úloha č.2:** Vyrobte stránku, ktorá cez dva `for` cykly vyrobí tabuľku $8 \times 8$ políčok a ako na šachovnici dá do čiernych políčok hviezdičky.

**Úloha č.3:** Vyrobte stránku, ktorá v PHPčku vypočíta najmenšie číslo, ktoré sa dá dvoma rôznymi spôsobmi napísať ako súčet dvoch tretích mocnín prirodzených čísel.

8. lekcia

# Formuláre
# alebo „Vstupné chlieviky“

V predošlej lekcii sme vytvorili nejaké PHP skripty. Veci, ktoré robili, boli síce relatívne zaujímavé, ale interaktivitou to práve neprekypovalo. Skript síce niečo do stránky vypísal, ale jediná vec, ktorú s tým môžeme urobiť je pokochať sa. Nemôžeme nijako ovplyvniť, čo skript zobrazí.

To sa samozrejme dá zmeniť. Už samotné HTML majú v sebe vec, ktorá sa volá formulár a ktorá slúži na to, aby sme mohli zadať nejaký vstup. Napríklad ak potrebujeme Googlu povedať, že čo nám má vyhľadať, tak ten rámik, do ktorého to píšeme je práve formulár.

Ako taký formulár môže vyzerať? Pozrite si nasledujúci HTML kód:

```
<html>
   <head>
      <title>Kalkulačka</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf8">
   </head>
   <body>
      <form action="vysledok.php" method="get">
         <table>
            <tr>
               <td>Zadaj číslo:</td>
               <td><input name="prvecislo" type="text" maxlength="8" /></td>
            </tr>
            <tr>
               <td>Zadaj ďalšie:</td>
               <td><input name="druhecislo" type="text" maxlength="8" /></td>
            </tr>
            <tr>
               <td>A čo s nimi:</td>
               <td>
                  <input name="operacia" type="radio" value="+" checked="checked" /> + <br />
                  <input name="operacia" type="radio" value="-" /> - <br />
                  <input name="operacia" type="radio" value="*" /> * <br />
                  <input name="operacia" type="radio" value="/" /> / <br />
               </td>
            </tr>
            <tr>
               <td><input type="submit" value="A teraz to zrátaj" /></td>
            </tr>
         </table>
      </form>
   </body>
</html>
```

Celý formulár sa nachádza medzi značkami `<form>` a `</form>`. Značka `form` má nastavené dva atribúty. Prvý je atribút `action` a označuje, ktorý skript bude dáta zozbierané formulárom spracovávať. Druhý je atribút `method`, ktorý označuje, akým spôsobom sa skriptu dáta pošlú. V prípade, že zvolíte metódu `get`, dáta sa pošlú ako súčasť URL. To môže byť užitočné pri ladení, ale môže to pôsobiť rušivo. Ak zvolíte metódu `post`, dáta sa odošlú tak, že to nebude v URL vidieť.

Vo vnútri `<form>` sa nedeje nič veľmi nezvyklého. Dali sme tam jednu tabuľku –

čisto z toho dôvodu, aby sme to mohli krajšie sformátovať, keby sa nám dodatočne chcelo ešte s tým hrať. Jediná nezvyklá vec sú značky `<input>`, ktoré reprezentujú práve tie miesta, kde je možné zadať používateľský vstup. Každý `input` má niekoľko atribútov. Atribút `type` hovorí, akého typu bude jednotlivý vstup. Máme tam tri typy. Typ `text` znamená, že sa zobrazí rámik, do ktorého sa dá písať. Typ `radio` znamená prepínač ako na starých rádiách. Je tam skrátka niekoľko možností, ale zapnutá môže byť len jedna z nich. Typ `submit` je tlačidlo, ktorého stlačenie spôsobí, že sa celý formulár odošle. Takých typov existuje viacero. Spomeňme typ `password`, ktorý funguje rovnako ako text, akurát, že vypisuje miesto jednotlivých písmeniek hviezdičky, typ `checkbox`, ktorý vyrobí zaškrtávacie okienko, typ `reset`, ktorý vyrobí tlačidlo na znovunastavenie pôvodných hodnôt alebo typ `file`, ktorý umožňuje zaslať serveru súbor.

Každý `input` okrem posledného (tlačidla `submit`) má aj atribút `name`. Tento atribút je dôležitý najmä pre skript, ktorý bude dáta spracovávať, pretože podľa neho sa bude určovať, ktoré je ktoré. Je dôležitý aj pre vstupy typu `radio`, pretože určuje, ktoré tlačidlá typu `radio` patria ku sebe (všetky také tlačidlá majú atribút `name` rovnaký) a keď sa niektoré z nich zapne, ostatné tlačidlá zo skupiny sa vypnú.

Textové polia v našom kóde má nastavený aj atribút `maxlength`, ktorým sa nastavuje, koľko maximálne znakov sa do poľa dá napísať. Prvé z rádiových tlačidiel má nastavený atribút `checked` na `checked` – spôsobí to, že práve toto tlačidlo bude na začiatku vybraté. Ten výber môže samozrejme používateľ zmeniť tak, ako chce.

Takže prvé dve úlohy:

**Úloha č.1:** Vyskúšajte si tu stránku napísať a pozrite si ju v nejakom prehliadači. Skúste zadať nejaké hodnoty a odoslať. Samozrejme to ešte nebude fungovať, pretože skript `vysledok.php` sme ešte nenapísali, ale pozrite sa, ako sa mení URL v závislosti na parametroch. Skúste metódu odosielania zmeniť z `get` na `post`. Čo sa zmení?

**Úloha č.2:** Nájdite na internete úplný zoznam typov, ktoré môže mať `input`. Zistite si, ako sa používajú a vyskúšajte si, ako fungujú.

Dobre. Takže vstupný formulár by bol. Teraz ešte treba napísať ten skript, ktorý vstup od používateľa vyhodnotí. Môže vyzerať napríklad takto:

```
<html>
   <head>
      <title>Kalkulacka</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf8">
   </head>
   <body>
   <?php
      $a = $_GET['prvecislo'];
      $b = $_GET['druhecislo'];
      $op = $_GET['operacia'];
      if ($op == "+")
      {
         $c = $a + $b;
         echo $a." + ".$b." = ".$c."<br \><br \> \n";
      }
      if ($op == "-")
```

```
        {
           $c = $a - $b;
           echo $a." - ".$b." = ".$c."<br \><br \> \n";
        }
        if ($op == "*")
        {
           $c = $a * $b;
           echo $a." * ".$b." = ".$c."<br \><br \> \n";
        }
        if ($op == "/")
        {
           $c = $a / $b;
           echo $a." / ".$b." = ".$c."<br \><br \> \n";
        }
     ?>
     <a href="formular.html">Späť</a>
    </body>
</html>
```

Na začiatku skriptu sme do premenných `$a`, `$b` a `$op` načítali hodnoty, ktoré nám poslal formulár. Tie hodnoty sme zobrali zo systémového asociatívneho poľa, ktoré sa nazýva `$_GET` a namiesto indexu sme uviedli meno patričného `input`-u. Keby skript získal premenné metódou `post`, bolo by ich treba hľadať v asociatívnom poli `$_POST`. No a potom sme už len zistili, akú operáciu používateľ zvolil, vypočítali a vypísali výsledok.

**Úloha č.3:** Vyskúšajte si to.

Vyzerá to dobre. Človek vyskúša pár vecí a ono to funguje. Lenže… Samozrejme sa nejaké lenže vyskytne. Skúste, čo to spraví, ak do nejakej kolónky napíšete nejaký nečíselný nezmysel. Alebo čo to spraví, ak skúsite vydeliť nulou. Môžete sa potom dozvedieť, že `jeden + jeden = 0` alebo dokonca môžete prísť k nejakému radostnému chybovému výpisu. V každom prípade sme sa hneď pri prvom príklade dotkli témy, ktorá sa časom ukáže ako nečakane dôležitá. Totiž – používatelia sú v lepšom prípade magori a v horšom zákeráci a sú schopní poslať nášmu skriptu také dáta, o ktorých sa nám ani nesnívalo – a napodiv sa vôbec nemusia nechať obmedzovať vstupným formulárom, pri metóde `get` stačí správne napísať URL a pri metóde `post` to tiež ide zariadiť. Takže je dôležité všetky dáta náležite ošetriť, aby nerobili šarapatu.[15]

---

15 Tento príklad je veľmi jednoduchý a nevinný. Zle napísaný skript, ktorý neošetrí vstupy, môže ale prípadnému útočníkovi vydať na milosť a nemilosť celú databázu. O tom, ako sa v PHP používajú databázy, si povieme neskôr, teraz len malá ukážka na vysvetlenie princípu a rizík s ním spojených. Predstavte si napríklad, že vo svojom skripte máte vytvorený nasledujúci SQL príkaz:

```
SELECT volaco FROM tabulka WHERE nieco_ine = '$pole';
```

kde premenná `$pole` je naplnená z formulára. Ak by sme vstup z formulára nijak neošetrili, niekto šikovný by nám do premennej `$pole` mohol vpašovať takúto hodnotu: `hihi'; DROP TABLE tabulka; --` Náš príkaz sa tým pádom zmení na takýto:

```
SELECT volaco FROM tabulka WHERE nieco_ine = 'hihi'; DROP TABLE tabulka; --'
```

To znamená „nájdi v tabuľke `tabulka` reťazec `hihi`, potom tabuľku `tabulka` zmaž“. Všetko za dvoma pomlčkami je komentár. Dokonalé, syntakticky správne a vražedné.

Takýmto radostiam je možné zabrániť starostlivým ošetrením premenných. Pri vstupných reťazcoch treba namiesto jednoduchého `$pole = $_GET['vstup'];` použiť `$pole = addslashes($_GET['vstup']);` Funkcia `addslashes` pridá pred špeciálne znaky (ako napríklad apostrof) spätné lomítka, takže v databázových

Ako sa to dá spraviť? Prvá vec, ktorú môžeme urobiť, je pri čítaní dát z poľa `$_GET` zavolať funkciu `doubleval`, ktorá akúkoľvek hodnotu skonvertuje na typ `double`. Takže patričné dva riadky budú vyzerať takto:

```
$a = doubleval($_GET['prvecislo']);
$b = doubleval($_GET['druhecislo']);
```

Ošetriť delenie nulou je tiež je jednoduché. Dá sa to napríklad takto:

```
if ($op == "/")
{
   if ($b == 0)
      echo "Nulou nedelíme!!!<br \><br \>\n";
   else
   {
      $c = $a / $b;
      echo $a." / ".$b." = ".$c."<br \><br \> \n";
   }
}
```

**Úloha č.3:** Upravte a vyskúšajte.

**Úloha č.4:** Skúste vymyslieť príkazový riadok, ktorý zadáte do okienka internetového prehliadača a ktorý vášmu skriptu metódou `GET` pošle ako operáciu %.

**Úloha č.5:** Prerobte to celé tak, aby sa dáta posielali metódou `post`.

**Úloha č.6:** Dorobte do formuláru kolónku na heslo a zariaďte to tak, aby kalkulačka kalkulovala iba vtedy, keď je heslo správne.

príkazoch to nerobí šarapatu. Takto upravený reťazec môžete v databázach používať bezpečne. Keď ho potrebujete potom znovu vypísať, spätné lomítka odstránite funkciou `stripslashes`. Pre vyššiu bezpečnosť je možné nastaviť webserver, aby to robil automaticky – aspoň pre Apache je to štandardné nastavenie.

9. lekcia

# Špecialitky
# alebo „Čo v C-čku nenájdete“

Táto lekcia je venovaná špecialitkám, ktoré v C-čku nenájdete, ale v PHP-čku sú a môžete ich s úspechom využiť.

Prvá šikovná vec, ktorá sa môže hodiť, je asociatívne pole. Je to pole, ktoré ako indexy nepoužíva čísla, ale hocičo, čo si zmyslíte. Keď ho vytvárate, môžete[16] na začiatok naznačiť, že pôjde o asociatívne pole, napríklad zadať doň jeden prvok: `$pole = array('odpoved' => 42);` Od tejto chvíle môžete s poľom `$pole` robiť akékoľvek psie kusy a ono to zvládne. Môžete spraviť napríklad priradenie `$pole['OdmocninaZDvoch'] = 1.414213562373095;` alebo napríklad `$pole[4321] = 'Buhaha';` Akurát ak použijete ako index číslo, ktoré nie je celé, tak sa zaokrúhli. Samozrejme k takýmto prvkom môžete pristupovať ako uznáte za vhodné, napríklad takto: `echo 'Odpoved na zasadnu otazku je'.$pole['odpoved'];`

Ak chcete niečo vykonať s každým prvkom asociatívneho poľa, môžete to vykonať pomocou nového typu cyklu nazvaného `foreach`. Pozrite si nasledujúci príklad:

```
<html>
   <head>
      <title>Asociativne pole</title>
   </head>
   <body>
      <table border="2">
         <?php
            $zoznam = array('Jano'=>'31417926');
            $zoznam['Fero'] = '27182818';
            $zoznam['Gerda'] = '10101010';
            $zoznam[19.22] = 3.14;
            foreach($zoznam as $meno => $cislo)
               echo "<tr><td>".$meno."</td><td>".$cislo."</td></tr>\n";
         ?>
      </table>
   </body>
</html>
```

Za `foreach` sa do zátvorky napísalo meno použitého asociatívneho poľa a za slovko `as` sa napísala premenná, do ktorej sa budú ukladať jednotlivé indexy (v našom prípade `$meno`). Potom sa napísal znak `=>` a po ňom meno premennej, do ktorej sa budú ukladať patričné hodnoty. Cyklus postupne prelezie celé pole a do dvojice `$meno`, `$cislo` dá všetky indexy a hodnoty poľa.

Uvedený cyklus sa dá napísať aj s pomocou klasického `while`. Vtedy ale budeme potrebovať funkciu `each`, ktorá vráti buď ďalšiu dvojicu kľúč-hodnota v poradí, alebo hodnotu `FALSE`, ak je už koniec poľa. Takže cyklus z predošlého príkladu sa dá napísať aj takto:

16 Nie je to ale povinnosť. Môžete rovno napísať `$pole['odpoved'] = 42;`

```
while( list($meno, $cislo) = each($zoznam))
      echo "<tr><td>".$meno."</td><td>".$cislo."</td></tr>\n";
```

Asi ste si všimli pomerne divoké priradenie `list($meno, $cislo) = each ($zoznam)`. V jazyku C by vám priradenie do nejakej funkcie celkom isto neprešlo. V PHP-čku sa zoberie dvojica zo zoznamu, zistí sa, do čoho sa to má priradiť, uvidí sa, že je to tiež dvojica, tak sa to pekne po zložkách rozbije a dá do správnych premenných.[17]

**Úloha č.1:** Vyskúšajte obidve varianty.

Ďalšia príjemná vec, ktorú ste si už určite všimli, je práca s reťazcami. Reťazce sa dajú jednoducho priradzovať do premenných, spájať s pomocou operátora `.` a akékoľvek veci sa na reťazce dajú jednoducho zmeniť. Všetko to ste už zažili. Okrem toho fungujú reťazcové funkcie z C-čka. A okrem nich ešte nejaké špecialitky. Za všetky spomeňme aspoň funkciu `explode`. Predstavte si, že máte nejaký reťazec (napríklad `"www.smnd.sk"`) a chceme ho rozdeliť na jednotlivé časti, ktoré sú oddelené bodkami. Stačí napísať

```
$pole = explode('.',"www.smnd.sk");
```

a v poli budete mať jednotlivé podreťazce. Teda `$pole[0]` bude `www`, `$pole[1]` bude `smnd` a `$pole[2]` bude `sk`.

Použitie je rôznorodé. Predstavte si, že by ste chceli vyrobiť skript, ktorý vypíše všetkých používateľov unixového systému.[18] Všetci používatelia sú uložení v systémovom súbore `/etc/passwd`. Pozrite si ho. Zistíte, že každému používateľovi je vyhradený jeden riadok. V tom riadku je viacero údajov, ako napríklad login, domovský adresár alebo číslo jeho skupiny. Medzi iným je tam uložené aj jeho meno. Jednotlivé informácie sú oddelené dvojbodkami. Meno je vždy piata informácia od začiatku. No a tieto informácie už stačia na to, aby sme mohli vytvoriť nasledujúci skript:

```
<html>
   <head>
      <title>Pouzivatelia</title>
   </head>
   <body>
      <?php
         $passwd = file("/etc/passwd");
         foreach($passwd as $line)
         {
             $kolonky = explode(":",$line);
             echo $kolonky[4]."<br>\n";
         }
      ?>
   </body>
```

17 V PHP to tiež nejde vždy, ale funkcia `list` je svetlá výnimka.

18 Taký skript o chvíľu napíšeme. Po vyskúšaní ho ale zo svojho adresára odstráňte. Zoznam používateľov by mohol uľahčiť prácu niekomu zvonku, kto by sa chcel do systému nelegálne votrieť.

```
</html>
```

Najprv nám funkcia `file` prečíta celý súbor a jeho riadky hodí ako položky do poľa `$passwd`. Cyklom `foreach` prejdeme celé pole (jednotlivé riadky budeme postupne ukladať do premennej `$line`. Funkciou `explode` riadok rozbijeme na jednotlivé informácie, ktoré dáme do poľa `$kolonky` a piatu z nich (to je tá s indexom 4 – pole je číslované od nuly) vypíšeme.

**Úloha č.2:** Vyskúšajte. Upravte skript tak, aby nevypisoval rôznych systémových používateľov, ale iba reálnych ľudí. (Poznámka: reálni používatelia majú ID používateľa – to je to číslo za druhou dvojbodkou – 500 alebo viac.) Keď to odladíte, nezabudnite skript presunúť z vášho adresára `www` niekam inam, nech nám sem zvonku nepozerajú, že akých máme používateľov.

Hlavné morálne poučenie z tejto lekcie je, že PHP obsahuje rôzne šikovné funkcie a je dobré o nich vedieť. A najlepší spôsob, ako sa o nich dozvedieť je pozrieť si dokumentáciu na http://www.php.net. Časť z nej je dokonca preložená do slovenčiny. Je tam podrobne popísaný jazyk, je množstvo rôznych kategórií funkcií, v ktorých sa oplatí pohrabať, ak chcete robiť niečo konkrétne (možno to urobil už niekto pred vami) a vôbec je to užitočný zdroj informácií.

**Úloha č.3:** Pozrite si, aké funkcie na prácu s reťazcami, na prácu s poľami a na prácu so súbormi má PHP.

10. lekcia

# SQL a PHP
# alebo „Pripojme si databázu“

Áno, v tejto lekcii dáme dohromady veci, ktorými sme sa doteraz v našom kurze zaoberali. Naučíme sa, ako z PHP kódu pripojiť databázu, ako zadávať SQL príkazy a ako zistiť, čo to vlastne spravilo. Ukážeme na začiatok krátky program, na ktorom vysvetlíme, ako sa veci vlastne robia:

```
<html>
   <head>
      <title>Pripojenie k databaze</title>
   </head>
   <body>
      <?php
         @ $db = mysql_pconnect("artus","guest","heslo");
         if (!$db)
         {
            echo "Je tam nejaky zadrhel. Neda sa pripojit k databaze.<br />\n";
            exit;
         }
         mysql_select_db("guest_common");
         $meno = "Henrich 8.";
         $prikaz = "INSERT INTO autori VALUES (NULL, \"".$meno."\");";
         $vysledok = mysql_query($prikaz);
         if ($vysledok)
            echo "Pocet vlozenych autorov: ".mysql_affected_rows()."<br />\n";
         else
         {
            echo "Ups, nepodarilo sa...<br />\n";
            echo "Chyba: ".mysql_error()."<br />\n";
         }
      ?>
   </body>
</html>
```

Samotný PHP kód začína príkazom `mysql_pconnect`. Tento príkaz vytvorí permanentné pripojenie k databáze. Je to lepšie, ako neustále pripájať a odpájať databázu, lebo sa tak šetrí čas a systémové prostriedky. Iste ste si všimli, že vstupné parametre sú meno počítača, meno a heslo. Je nepríjemné, že heslo sa tam musí vyskytovať. Je preto dôležité na operačných systémoch, kde je to možné (čo už sú dnes takmer všetky) zabezpečiť, aby súbory so skriptami neboli verejne čitateľné.

Novinkou je aj ten zavináč (znak @) pred samotným príkazom. Ten spôsobí, že ak sa nepodarí pripojiť k databáze, PHP nevygeneruje žiadny chybový výpis. Ten sme si vyrobili vo vlastnej réžii sami.

Konkrétnu databázu si vyberieme príkazom `mysql_select_db`. Ako parameter zadáme jej meno.

Poďme teraz zadať nejaký SQL príkaz. Pokúsime sa medzi autorov zaradiť monarchu Henricha VIII. Mohli by sme to spraviť napríklad nasledujúcim príkazom:

```
mysql_query("INSERT INTO autori VALUES (NULL, \"Henrich 8.\")");
```

V našom príklade sme sa ale rozhodli uložiť meno najprv do premennej `$meno` a potom aj celý príkaz vložiť do premennej `$prikaz`. Je tak totiž vidno, že príkazy nemusia byť statické a že môžu záležať napríklad od toho, aký vstup zadá používateľ. Všimnite si, že keď sme chceli, aby boli úvodzovky súčasťou reťazca, museli sme pred ne dať spätné lomítko.

Ak sa príkaz `mysql_query` nepodarí vykonať, vráti hodnotu `FALSE`. Môžeme teda skontrolovať, či je všetko tak, ako má byť. Ak sa príkaz vykonať podarilo, môžeme používateľa obšťastniť výpisom, koľko že riadkov sme vsunuli. Ak sa príkaz vykonať nepodarí, funkcia `mysql_error()` nám vypíše podrobnejšiu správu o chybe. Žiaľ v angličtine.

**Úloha č.1:** Vyskúšajte.

SQL príkazy zadávať vieme. Ako vždy to má ale jeden háčik. Keď vykonáme nejaký `SELECT`, ten vráti výsledky. Ako sa ale k tým výsledkom dostať? Takto:

```
<html>
   <head>
      <title>Select</title>
   </head>
   <body>
      <?php
         @ $db = mysql_pconnect("artus","guest","heslo");
         if (!$db)
         {
            echo "Je tam nejaky zadrhel. Neda sa pripojit k databaze.<br />\n";
            exit;
         }
         mysql_select_db("guest_common");
         $prikaz = "SELECT autori.meno,knihy.nazov FROM autori,knihy
                    WHERE knihy.autor_id = autori.autor_id;";
         $vysledok = mysql_query($prikaz);
         if (!$vysledok)
         {
            echo "Nieco sa zopsulo: ".mysql_error()."<br />\n";
            exit;
         }
         $pocetriadkov = mysql_num_rows($vysledok);
         echo "<table border=\"2\">\n";
         for ($i = 0; $i < $pocetriadkov; $i++)
         {
            $riadok = mysql_fetch_array($vysledok);
            echo "<tr><td>".$riadok["meno"]."</td><td>".
                          $riadok["nazov"]."</td></tr>\n";
         }
         echo "</table>\n";
      ?>
   </body>
</html>
```

Pripojíme sa k databáze rovnako ako v predošlom príklade. Vyrobíme si `SELECT`,

ktorý nám vypíše všetky knižky aj s autormi, uložíme si ho do premennej `$prikaz` a zavoláme `$mysql_query`. Ošetríme chyby a môžeme sa pustiť do vypisovania.

Funkcia `mysql_num_rows` zistí, koľko riadkov `SELECT` vrátil. Toľkokrát potom necháme zbehnúť cyklus `for`. Zakaždým zavoláme funkciu `mysql_fetch_array`, ktorá nám vráti ďalší riadok výsledku ako asociatívne pole. Pole vložíme do premennej `$riadok`. Jednotlivé položky budú `$riadok["meno"]` a `$riadok["nazov"]`. Vypíšeme ich do tabuľky.

**Úloha č.2:** Vyskúšajte.

**Úloha č.3:** Vyrobte HTML formulár, ktorý načíta meno autora a pošle ho skriptu, ktorý z databázy vypíše názvy všetkých kníh, ktoré daný autor napísal.

**Úloha č.4:** (pre expertov) Upravte formulár z úlohy č.3 tak, aby v ňom nebolo vstupné textové pole, ale combo box, v ktorom sa budú nachádzať všetky prípustné mená autorov zoradené podľa abecedy. (Na jeho vytvorenie budete musieť použiť PHP skript, ktorý mená prečíta z databázy.) Detaily ohľadom vytvárania combo boxov si zistite na internete.

11. lekcia

# Sedenia
# alebo „Pusti ma, to som ja“

Pravdepodobne ste už na webe narazili na miesta, kde sa treba prihlásiť, aby ste si mohli prezerať obsah, či vykonávať nejakú ďalšiu činnosť. Či už ide o elektronický obchod, diskusné fórum alebo web lokálnej komunity. Podivné je na tom to, že keď sa prihlásite na jednej stránke, mnoho ďalších stránok vie, že ste to vy. Jednotlivé stránky si medzi sebou musia odovzdávať vašu identifikáciu, aby bolo jasné, že niekto iný vo vašom mene nevykonáva niečo nekalé.

Keby ste takýto systém mali naprogramovať sami, mohli by ste sa do toho pustiť viacerými spôsobmi. Jedna z možností je pri každom volaní novej stránky odovzdať s pomocou metódy `GET` alebo `POST` vašu identifikáciu a na začiatku každej stránky overiť v nejakej tabuľke databázy, či ste to naozaj vy. Iná možnosť je použiť cookies – informácie, ktoré si server môže uložiť na strane klienta a potom ich využiť. Na stránku potom nepustíme nikoho, kto správnu hodnotu nemá nastavenú. Úplne najjednoduchšie ale bude použiť správu sedení, ktorá je priamo vstavaná v jazyku PHP. Táto samozrejme používa niektorú z uvedených metód, ale dá sa použiť bez toho, aby ste sa s nimi potrebovali babrať.[19]

Mechanizmus, ktorý umožňuje odovzdávať identifikáciu (alebo aj iné premenné) medzi jednotlivými stránkami sa nazýva sedenie (po anglicky session). Sedenie po inicializácii môže obsahovať premenné, ktoré sú pre neho špecifické a ktoré sú uložené v asociatívnom poli `$_SESSION`. Ak stránka otvorí sedenie, môže do tohto poľa premenné pridávať, čítať ich alebo ich mazať. Sedenie skončí buďto zrušením sedenia (keď sa používateľ odhlási), alebo vypršaním časového limitu (keď používateľ dlho nič nespraví a systém usúdi, že od počítača odišiel). Veľkosť časového limitu sa nastavuje v konfiguračnom súbore pre PHP (v prípade UNIXových systémov súbor `php.ini` v adresári `/etc`).

Ako to funguje, si môžete pozrieť v nasledujúcom príklade, ktorý bude trocha dlhší. Skladá sa z troch súborov. Prvý (`index.php`) má na starosti prihlásenie:

```
<?php
   session_start();
?>

<html>
     <head>
          <title>Prihlasenie</title>
     </head>
     <body>
          <?php
               if (! isset($_POST['meno']) && !isset($_SESSION['User']) )
                      /*este nic nezadali */
               {
```

19 To samozrejme neznamená, že nie je užitočné vedieť, ako také cookies fungujú. Vrelo odporúčame naučiť sa to.

```
        ?>
        <form method="post" action="index.php">
        <table>
        <tr><td>Login:</td>
        <td><input type="text" name="meno"></td></tr>
        <tr><td>Password:</td>
        <td><input type="password" name="heslo"></td></tr>
        </table>
        <input type="submit" value="Prihlasit">
        </form>
        <?php

                }
                else if (isset($_POST['meno']))
                       /* zistime, ci dali dobre heslo */
                {
                    $meno = addslashes($_POST['meno']);
                    $heslo = addslashes($_POST['heslo']);
                       $hash = md5($heslo);
                    $db_conn = mysql_connect('localhost','guest','heslo')
                         or die('Neda sa pripojit k SQL serveru');
                    mysql_select_db('guest_common',$db_conn)
                         or die('Neda sa pripojit k databaze');
                    $query = 'SELECT * FROM passwd '
                                ."WHERE login='$meno' "
                                ."AND password='$hash'";
                    $vysledok = mysql_query($query, $db_conn);
                    if (mysql_num_rows($vysledok) > 0)
                         /* niekto taky v databaze je */
                         $_SESSION['User'] = $meno;
                    else
                         /* nikoho sme nenasli */
                         echo "Nepozname vas. Chodte inde.<br>\n";
                }
            if (isset($_SESSION['User']))
                echo $_SESSION['User']."., vitajte v systeme.<br>\n";
         ?>
         <a href="tajne.php">Tajny dokument</a><br>
         <a href="logout.php">Odhlasit</a>
       </body>
    </html>
```

Úplne na začiatku sa zavolá funkcia `session_start()`. V prípade, že je sedenie spravované s pomocou cookies, je nutné túto funkciu zavolať ešte predtým, než sa odošle akýkoľvek výstup, teda ešte pred uvedením značky `<html>`. Táto funkcia zabezpečí, aby sa medzi serverom a browserom vymenili všetky dôležité dáta.

V prípade, že nie je nastavená premenná sedenia `$_SESSION['User']` (ktorú nastavíme až vtedy, keď sa podarí úspešné pripojenie) ani sa nikto práve nepokúša prihlásiť (nie je nastavená premenná `$_POST['meno']`), zobrazí sa prihlasovací formulár, ktorý si pýta meno a heslo. Ak je premenná `$_POST['meno']` určená, predpokladáme, že používateľ sa práve snaží prihlásiť. Skontrolujeme, či je taký používateľ u nás v databáze v tabuľke `passwd` a či md5-súčet jeho hesla uložený v premennej `$hash` zodpovedá tomu, ktoré máme uložené v tabuľke.[20] Ak teda

20 Z bezpečnostných dôvodov nie je rozumné uchovávať v databáze nezašifrované heslá

`SELECT` nájde aspoň jeden riadok, ktorý zadaným údajom zodpovedá, používateľ je úspešne prihlásený. Sedeniu to dáme vedieť tak, že jeho premennú `$_SESSION['User']` nastavíme na meno prihlasujúceho sa používateľa. Ak nikoho takého v databáze nenájdeme, vypíšeme odrádzajúci výpis.

Ak je hodnota `$_SESSION['User']` nastavená, vypíše sa privítacia veta. Na záver skriptu sa vypíšu dve linky na tajný dokument, ktorého obsah by nemal vidieť nikto neprihlásený a na odhlásenie.

Tajný dokument `tajne.php` môže vyzerať napríklad takto:

```
<?php
   session_start();
?>
<html>
   <head>
      <title>Tajny dokument</title>
   </head>
   <body>
      <?php
         if (! isset($_SESSION['User']))
         {
            echo "Nelezte sem, sem mozu iba prihlaseni<br>\n";
            echo "</body>\n</html>\n";
            exit;
         }
      ?>
      <b>Tajna sprava:</b> Vsetko je inak.<br>
      <a href="index.php">Domov</a><br>
      <a href="logout.php">Odhlasit</a>
   </body>
</html>
```

Opäť na začiatku zavoláme funkciu `session_start()`, ktorá nám zabezpečí inicializáciu poľa `$_SESSION`. Ak v ňom nie je nastavená premenná `$_SESSION['User']`, znamená to, že používateľ sa zatiaľ neprihlásil, odkážeme ho teda do patričných medzí a skončíme. V prípade, že premenná nastavená je, používateľ bol riadne prihlásený a vypíše sa mu tajná správa.

Odhlásenie zo systému sa môže previesť nasledujúcim spôsobom (súbor `logout.php`):

```
<?php
   session_start();
?>
<html>
   <head>
      <title>Odhlasenie</title>
   </head>
   <body>
      <?php
         if (! isset($_SESSION['User']))
         {
            echo "Naco sa odhlasujete, ked ste sa neprihlasili.<br>\n";
            echo "</body>\n</html>\n";
            exit;
```

```
            }
            echo $_SESSION['User'].", majte sa pekne, logout.<br>\n";
            unset($_SESSION['User']);
            session_destroy();
         ?>
         <a href="index.php">Znovu prihlasit</a><br>
         <a href="tajne.php">Tajny dokument</a><br>
      </body>
   </html>
```

Opäť na začiatku naštartujeme sedenie. Skontrolujeme, či je vôbec koho odhlasovať. Ak nie je, podáme o tom správu. Ak je niekto prihlásený, slušne sa rozlúčime, vymažeme premennú `$_SESSION['User']` a zavoláme funkciu `session_destroy()`, ktorá sedenie ukončí.

**Úloha č.1:** (Tradičná.) Vyskúšajte.

**Úloha č.2:** Dorobte k tomu prihlasovací formulár, ktorým sa do systému môžu prihlasovať noví používatelia. Heslo načítajte dvakrát a porovnajte, či používateľ zadal dvakrát tú istú hodnotu. Ak nie, vypíšte chybovú správu, ak áno, zapíšte meno a md5 súčet hesla do tabuľky v databáze.

**Úloha č.3:** Pridajte do tabuľky v databáze ďalší stĺpec, ktorý obsahuje informáciu, či je daný používateľ správca, alebo nie. Informáciu o tom uložte do premennej poľa `$_SESSION` a správcovi pri výpise tajných stránok a pri odhlasovaní preukážte patričnú úctu. Pri odhlasovaní nezabudnite zmazať aj túto premennú.

11. lekcia

# Plánujeme rozhranie alebo „Čo všetko bude treba urobiť“

V predošlých lekciách ste pochytili čo to o databázach a SQL jazyku a videli ste približne aj ako sa robia internetové stránky s pomocou jazyka PHP. V každom prípade už viete dosť na to, aby ste sa mohli pustiť do rozsiahlejšieho projektu.

Pamätáte sa ešte, ako sme kedysi v štvrtej lekcii navrhovali databázu pre knižnicu? Samotná databáza bola navrhnutá dobre. Naučiť ale knižničné tetušky SQL jazyk, aby v ňom vyhľadávali, menili záznamy a zachovali pritom databázu v konzistentnom stave je ale pravdepodobne úloha nad ľudské sily.[21] Preto treba vyrobiť nejaké klikateľné rozhranie, ktoré by sa relatívne príjemne ovládalo, dbalo by na to, aby používateľ nezadával hlúposti, kontrolovalo ho a nedopustilo, aby v databáze spáchal nejaké zmätky. Návrh takéhoto rozhrania nie je v žiadnom prípade jednoduchá vec. V tejto lekcii sa budeme venovať tomu, ako taktéo rozhranie naplánovať.

Prvá fáza tvorby projektu pozostáva zo **špecifikácie požiadaviek zo strany zákazníka**. Zákazník spíše, čo chce, aby požadovaný systém bol schopný urobiť. V našom prípade by taká špecifikácia mohla vyzerať napríklad takto:

- V systéme budú tri typy úloh. Úlohy, ktoré vykonávajú pracovníci knižnice (PK), úlohy, ktoré vykonávajú klienti (K) a úlohy, ktoré vykonáva samotný systém (S). Pri každej úlohe bude uvedené, o aký typ sa jedná.
- (PK) Systém bude evidovať zoznam kníh
  - (PK) Ku každej knihe bude evidovať základné dáta
  - (PK,K) Ku každej knihe môže evidovať niekoľko komentárov
- (PK) Systém bude evidovať zoznam klientov
  - (PK) Ku každému klientovi bude evidovať základné dáta
  - (PK) Ku každému klientovi môže evidovať niekoľko poznámok
- (PK) Systém bude evidovať, ktorý klient si kedy ktorú knihu požičal a kedy ju vrátil
- (S) Systém zabezpečí mechanizmus upomienok
  - (S) Systém automaticky zašle upomienku po 1, 2 a 3 mesiacoch, potom vec postúpi právnemu oddeleniu knižnice
- (K,S) Systém zabezpečí mechanizmus objednávok a rezervácií
  - (K,S) Ak si klient objedná knihu, ktorá je k dispozícii, systém mu ju na v systéme nastavený čas zarezervuje
  - (K,S) Ak si klient objedná knihu, ktorá nie je k dispozícii, systém mu po jej vrátení odošle e-mail a zarezervuje mu ju na čas nastavený v systéme

21 Touto vetou sa nijako nechcem dotknúť knižničných tetušiek. Uchovať väčšiu databázu v konzistentnom stave, využívať ju a dohovárať sa s ňou iba prostredníctvom príkazového riadku je úloha nad sily bežného smrteľníka.

Na základe takto vypracovaných požiadaviek je možné vypracovať s tvorcom softvéru zmluvu o analýze a implementácii.

Ďalšou fázou tvorby projektu je **analýza**. Robí ju tvorca softvéru a konzultuje ju so zákazníkom. Analýza pozostáva v presnej špecifikácii objektov, ktoré sa v systéme vyskytujú a s procesmi, ktoré sa s nimi majú udiať. V našom prípade môže vyzerať takto:

**Knihy**

Súčasťou evidencie kníh je aj množina autorov, vydavateľstiev a kategórií.

O každej knihe systém eviduje nasledujúce dáta (všetky sú povinné):

- kniha_id (systémový id, používateľa sa netýka)
- nazov (ľubovoľný text)
- autor (väzba na jedného autora)
- vydavatel (väzba na vydavateľstvo)
- rok_vydania (ľubovoľný int)
- isbn (vo formáte x-y-z-q kde x,y,z,q sú integery)
- kategoria (väzba na kategóriu)
- stav (vymenované – jedna z možností)
- cena (väčšia alebo rovná 0)
- aktivna (áno/nie)
- datum vloženia do databázy

Vydavateľ, autor a kategória nemajú ďalšie špeciálne dáta, môžu byť zdieľané viacerými knihami. S ich zoznamami je možné vykonať nasledujúce operácie: pridať položku, zmeniť položku (položky sú zdieľané, zmenia sa pre všetky knihy, kde sú použité), zmazať položku (zmazať sa dá, len ak nebola použitá).

Pri vkladaní novej knihy alebo pri zmene existujúcej knihy užívateľ vyberá zo zoznamov autora, vydavateľa a kategóriu. Ak požadovaná položka neexistuje, užívateľ ju môže hneď pridať alebo meniť.

Pracovník knižnice môže knihu aktivovať/deaktivovať. Len aktívna kniha sa dá požičiavať. Deaktivovať sa dá aj požičaná kniha (v prípade straty).

Vymazať sa dá len kniha, ktorá nie je požičaná, ani nikdy nebola.

**Systém výpožičiek**

Knihu si môže naraz vypožičať len 1 zákazník

Štandardná doba výpožičky je 1 mesiac (nastavenie systému, dá sa zmeniť), pracovník knižnice ju môže zmeniť pre konkrétnu výpožičku

Keď zákazník knihu vráti, zapíše sa dátum vrátenia

Ak zákazník chce knihu dlhšie a oznámi to pred uplynutím výpožičnej doby, je možné štandardnú dobu výpožičky aj dodatočne zmeniť.

Ak už výpožičná doba uplynula a kniha nie je vrátená, zašle sa zákazníkovi upomienka.

**Pokuty**

Strata alebo poškodenie knihy

- PK zaeviduje, že kniha je stratená/poškodená
- zákazník zaplatí cenu knihy/cenu opravy (cenu opravy zadá PK, nie je nastavená v systéme)

Uplynutie výožičnej doby

- Ak zákazník knihu vráti (alebo oznámi, že knihu stratil) po uplynutí výpožičnej doby, zaplatí pokutu, ktorá závisí od počtu upomienok. Výška pokuty v závislosti od počtu upomienok je systémové nastavenie.
- Zaplatené pokuty systém neeviduje.

**Aplikačná logika**

**Funkcia: prehľad pohľadávok klienta**

Pohľadávky klienta sú:

- vypožičané knihy
- súčet ešte nezaplatených pokút (robí sa to len pre výpožičky v stave „n-tá upomienka“ )

**Funkcia: obsluha klienta**

PK vyberie klienta z databázy, alebo založí nového klienta. PK vyberie jednu z možných služieb:

- Založenie nového klienta. PK vyplní dáta klienta (všetky dáta sú povinné). Pri založení klienta je klient deaktivovaný.
- Aktivovanie klienta (robí PK)
- Deaktivovanie klienta
  - ak má klient požičanú knihu, nemožno ho deaktivoavať
- Zmazanie klienta
  - zmazať možno len toho klienta, ktorý je deaktivovaný a nikdy nemal nič požičané
- Vypožičanie knihy
  - vytvorenie záznamu o výpožičke výberom knihy zo zoznamu a vyplnením dát:
    - požadovaný dátum vrátenia (systém navrhne podľa štandardnej doby výpožičky a používateľ to môže zmeniť)
    - stav výpožičky a dátum požičania vyplní systém (zobrazí sa iba na čítanie)
- Vrátenie knihy
  - zadá sa dátum vrátenia

  - PK skontroluje reálny stav knihy (normálna, poškodená atď. a ak treba zmení stav knihy v systéme)
  - systém ukáže, či sú na túto výpožičku upomienky a ukáže výšku pokuty
- Oznámenie o strate knihy
  - PK zmení stav výpožičky na „nenávratná“
- Predĺženie výpožičnej doby
  - systém ukáže, či sú na túto výpožičku upomienky a ukáže výšku pokuty
  - PK zmení požadovaný dátum vrátenia. Pri zmene sa nastaví stav výpožičky na „normálna“

**Dáta výpožičiek**

Systém eviduje tieto dáta výpožičky:

- požičaná kniha (väzba)
- zákazník, ktorý si požičal knihu (väzba)
- dátum výpožičky
- dátum vrátenia
- stav výpožičky (viď nižšie)
- dátum poslednej zmeny stavu (Pozor! Toto v našom pôvodnom návrhu nebolo, ale mnohé veci to zjednoduší. Zmeňte tabuľku!)
- väzba na nula až veľa upomienok

**Stav výpožičky**

Výpožička môže mať tieto stavy:

- normálna (pred uplynutím výpožičnej doby)
- prvá upomienka
- druhá upomienka
- tretia upomienka
- u právnikov
- nenávratná

Stav výpožičky nastavuje systém. Ak systém nastaví stav výpožičky na „nenávratná“, tak nastaví stav knihy na deaktivovaná.

**Funkcia: zoznam kníh pre zákazníka**

Zákazník môže vidieť zoznam všetkých aktívnych kníh (aj vypožičaných). V zozname vidno tieto dáta:

- Názov knihy, autor, vydavateľ, kategória
- Stav výpožičky knihy pre zákazníka (či je kniha požičaná alebo nie, ale nevidno presne stav výpožičky – napr. ktorá upomienka)
- Ak je kniha požičaná, a stav výpožičky je „normálna“ , tak v zozname je stav = „požičaná“ a dátum, kedy má byť vrátená
- Ak je stav výpožičky iný ako „normálna“ tak v zozname je stav = „choď vynadať“ a meno a adresa a tel. číslo toho, kto ju má požičanú :-)

Zoznam kníh možno filtrovať podľa všetkých dát, ktoré vidno v zozname.

**Funkcia: zoznam kníh pre PK**

PK môže vidieť zoznam všetkých kníh. V zozname vidno tieto dáta:

- Názov knihy, autor, vydavateľ, kategória, stav, cena, rok vydania, ibsn (názov, rok vydania, stav, cena a ibsn je možné meniť, je možné meniť väzby na autora, vydavateľa a kategóriu, v prípade potreby je možné jednoducho nového autora, vydavateľa alebo kategóriu vytvoriť. Každú zmenu treba potvrdiť.)
- Stav výpožičky knihy pre personál (či je kniha požičaná alebo nie a ak áno, kto ju má požičanú a koľko upomienok už bolo vydaných)

Zoznam kníh možno filtrovať podľa všetkých dát, ktoré vidno v zozname.

**Funkcia: objednávka a rezervácia knihy**

Zákazník si vyhľadá knihu zo zoznamu všetkých aktívnych kníh (aj vypožičaných). Ak vyberie nepožičanú knihu systém mu ponúkne rezerváciu. Ak vyberie vypožičanú knihu, systém mu ponúkne objednávku.

Ponuka rezervácie obsahuje dátum dokedy je rezervovaná (systém má ten dátum zo systémových nastavení)

Pri objednávke nie sú žiadne ďalšie dáta. (Ak sa objednaná kniha vráti, automaticky sa vytvorí rezervácia pre prvého zákazníka, ktorý si knihu objednal a zákazníkovi sa odošle e-mail, ktorý obsahuje informáciu aká kniha a dokedy bude rezervovaná. Ak si ju včas nevyzdvihne, kniha bude automaticky rezervovaná ďalšiemu zákazníkovi, ktorý ju má objednanú.)

Zákazník potvrdí rezerváciu/objednávku, systém to zapíše do DB.

**Funkcia: generovanie upomienok**

Upomienka je generovaná pre každú relevantnú výpožičku osobitne (t.j. nie pre viacero naraz) Táto funkcia je „batch“, dávková úloha, ktorá sa bude spúšťať na pokyn personálu knižnice. Funkcie robí toto:

- Podľa aktuálneho dátumu, dátumov požadovaného vrátenia na výpožičkách, stavu výpožičiek, a systémového nastavenia sa aktualizuje stav výpožičky a dátum posledného stavu výpožičky
- Podľa nového stavu výpožičky sa generujú upomienky (zapíšu sa do DB). Majú stav „neposlaná“

**Funkcia: posielanie upomienok**

- Systémom hromadnej korešpondencie sa urobia listy (dokumenty upomienok) pre tie upomienky, pre ktoré je stav „neposlaná“
- Týmto upomienkam sa zmení stav na „poslaná“

**Dáta upomienky**

- dátum generovania
- väzba na výpožičku
- stav „poslaná“, „neposlaná“
- typ upomienky (1. , 2. , 3.)

**View: upomienky klienta**

Tento náhľad zosumarizuje neposlané upomienky pre jedného klienta. Bude tam pre každú výpožičku:

- Klient (väzba)
- Kniha (názov, autor, dátum výpožičky)
- Druh upomienky
- Suma pokuty za knihu (podľa číselníka pokút podľa druhu upomienky)

**View: celková suma klienta**

- Klient
- Súčet pokút

To by bola analýza projektu. Na jej základe sa dá napísať zmluva o realizácii projektu, ktorá už obsahuje konkrétne údaje o financiách. Počas samotného vývoja sa často stane, že buď zákazník, alebo vývojový tím príde na to, že v analýze ešte niečo dôležité chýba.[22] Takéto prídavky sa stávajú predmetom samostatného jednania, robí sa pre ne samostatná analýza a je pre ne nutné zvlášť dohadovať finančnú stránku. Zákazník má totiž často prístup „takú maličkosť nám predsa urobíte zadarmo“, niektoré „maličkosti“ sa ale môžu ukázať byť také náročné, že vyžadujú niekoľko človekodní práce a preto je starostlivá analýza nutná.

**Úloha č.1:** (dnes iba jedna!!!) Urobte to.

22 Možno ste si nevšimli, ale v našej analýze chýba akákoľvek správa komentárov ku knihám a zákazníkom. Ako cvičenie si môžete skúsiť túto analýzu dorobiť.