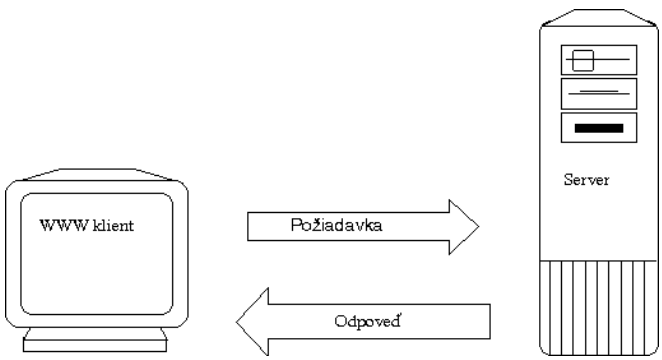
Malé veľké databázy II /1.časť

Asi ste si všimli malú zmenu názvu nášho seriálu. Začíname znovu od prvej časti, ale už vo vyššej triede. To preto, lebo už sme skúsení databázisti (a databázistky!). Vieme nainštalovať MySQL server, vieme vytvoriť databázu a v nej potrebné tabuľky. Tabuľky vieme naplniť rôznymi dátami rôznymi metódami. Vieme pracovať so záznamami všelijakými spôsobmi. Ba dokonca, vieme stanoviť, kto a čo môže robiť s tou - ktorou tabuľkou. Všetko je to skutočne fajn, len... Len to všetko doposiaľ dosahujeme iba z príkazového riadku. Teda, my sami, ako už znalí guruovia, by sme nevymenili príkazový riadok konzoly za žiadny (teda skoro žiadny) iný administrátorský nástroj, ale prenesme sa teraz duchom do našej imaginárnej knižnice. Presvedčíme našu slečnu referentku, aby si vyhľadávala záznamy v *mysql* monitore a ovládala pomerne zložitú syntaxiu príkazu ***SELECT***, nedajbože aj s určitými podmienkami? No ak by bola taký fanatik do počítadiel ako my, alebo by sme ju správne motivovali korunkami, tak možno. Ale musíme predpokladať, že do knižnice si prídu požičať knihy aj ľudia s veľmi malými znalosťami počítačov alebo dokonca počitačovo negramotní! Ako si budú vyhľadávať knihy v našom projekte? Aké teda majú byť naše projekty? Musia sa ľahko ovládať, najlepšie len jednoduchým klikaním myši alebo potvrdzovaním voľby jednou klávesou. Písať by sa malo len minimálne, a ak, tak len prihlasovacie *meno, heslo!!! a vyhľadávací reťazec*. Zobrazovaná informácia by mala mať určitú štábnu kultúru, nie zmäť znakov na stovkách riadkov. Takýmto projektom hovoríme user-friendly, teda užívateľsky priateľské. A v tom je to najväčšie programátorské čaro. Princípy a alogoritmy ovládame všetci. Ale v akej podobe ich podsunieme našim užívateľom, to už je ozajstný kumšt. Práve podľa prítulnosti prostredia projektu užívatelia delia projekty na dobré a zlé.
Každé umenie potrebuje dve základné veci: **technologické okolie** a **umelecký cit**. Umelecký cit si nosí každý v sebe. A technologické okolie si teraz vysvetlíme.

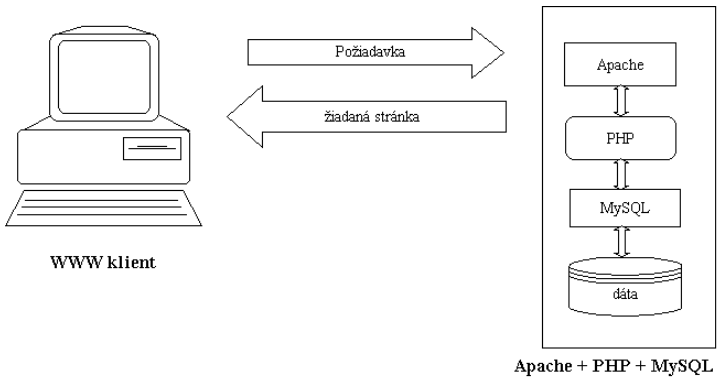
Troška teórie
Tak ako maliar obrazu okrem umeleckého citu potrebuje plátno, štetce a farby, aj programátor potrebuje k tvorbe projektu určité nástroje.
Ako hlavný nástroj databázového projektu môžeme považovať databázový stroj, čo je v našom prípade MySQL. No aby sme sa mohli odpútať od príkazového riadku, budeme ešte potrebovať niečo, čo bude spracovávať naše požiadavky na dáta a pripraví výsledok v žiadanej forme. Okrem toho ešte potrebujeme, aby sa už takto vytvorené stránky odoslali ku klientovi, teda nejaký http server.

Apache a PHP
Každý, kto už aspoň trochu pričuchol k internetu, sa s http serverom stretol. To je server, ktorý komunikuje s klientom pomocou ***hypertext transfer protokolu*** (http). Jeho úloha je to veľmi podobná ako nakupovanie v obchode cez pult - predavačke povieme svoju požiadavku, ona sa trochu pozvŕta medzi regálmi, a keď nami žiadaný tovar nájde, tak nám ho poskytne.
Na obr. č 11-1 je schématicky naznačená úloha takého http servera: prijať požiadavku na stránku od klienta a následne mu vrátiť požadovanú stránku.



Httpd démon, vytvorený v stredisku NCSA, prešiel, tak ako všetko vo svete softvéru, mnohými opravnými záplatami, angl. ***a patch*** (čítaj a pač). Podľa fonetického znenia záplat dostal svoje meno aj náš indián. Server ***Apache*** (čítaj apač) je voľne dostupný http server. Niekedy mu hovoríme aj web server. Slúži hlavne ako server HTML stránok vo svete Internetu, ale v poslednej dobe sa stáva veľmi obľúbeným aj v menších podnikových intranetoch. Preto ho aj my budeme používať. Povedali sme si, že samotný http server vie len prijať požiadavku

klienta na určitú stránku. Je to taký jednoduchý sluha (servo = sluha), ktorý ju vyhľadá, a ak ju má, pošle mu ju. Nič viac! Ak by sme po ňom chceli, aby našiel konkrétny záznam v našej databáze, neuspeli by sme. Preto potrebujeme akýsi „medzikus“ medzi web serverom a MySQL, ktorý by od apača prevzal požiadavku na zložitejšiu stránku, ktorú on sám nemá, spracoval ju, vytvoril určitú stránku a tú podsunul apačovi - reku, pošli to tomu klientovi. A takýto medzikus je ***PHP - Professional Home Pages***. Ako samotný názov naznačuje, pôjde o preprocesor, ktorý vie na základe určitých požadovaných kritérií vytvárať naozaj profesionálne domáce stránky. A to je to, čo by sme chceli dosiahnúť. Schématické znázornenie je na obr. č.11-2:



Ako to funguje?
Veľmi inteligentne. Ak klient pošle požiadavku na klasickú stránku v HTML formáte, server zistí, či takú stránku má. Ak áno, pošle ju www klientovi, ten ju spracuje a zobrazí na obrazovke počítača, ktorý o danú stránku žiadal.
Ak však požiadame o stránku s obsahom PHP príkazov, Apache odovzdá túto požiadavku preprocesoru PHP. Ten spracuje PHP príkazy a na ich základe dynamicky vygeneruje HTML stránku, ktorú posunie http serveru. Ten, ako dobrý služobník, pošle takto vytvorenú stránku klientovi.
A čo sa deje, ak klient potrebuje konkrétne údaje z databázy? No apač zistí, že klient požaduje stránku s obsahom PHP príkazov, tak ako sme si to už spomenuli vyššie. Preto odovzdá požiadavku PHP preprocesoru. Ten zistí, že PHP skript obsahuje príkazy, ktoré sa odvolávajú na prácu s databázovým strojom, preto sa preprocesor spojí s SQL serverom a odovzdá mu príkazy pre prácu s databázou, teda nám dobre známe SQL príkazy. SQL server (a teraz je to jedno, o ktorý sa jedná) spracuje tieto príkazy a výsledok svojej činnosti vráti PHP. PHP preprocesor získané hodnoty zapracuje do HTML stránky tak, aby to malo určitú štruktúru a túto stránku posunie web serveru. Nakoniec ju tento pošle žiadateľovi.
Aby to takto fungovalo, musíme mať nainštalovaný SQL server, http server a PHP. Všetky tri zložky musia byť navzájom logicky previazané. Toto sa dosahuje úpravou príslušných konfiguračných súborov. Tak ako sme si už zvykli, všetky tri súčasti, ale aj www klient môžu, ale nemusia bežať na jednom počítači.
Inštalovať a konfigurovať MySQL server už vieme. Apache a PHP sa naučíme teraz.
My si ukážeme spôsob inštalácie na platforme Window XX a na Linuxe. To, že sa môžeme pripájať z Windows na Linux a naopak, už ani hovoriť nemusím.

Inštalácia a kofigurácia Apache a PHP na platforme Windows
Keďže sa jedná o základy každého dobrého internetového či intranetového servera, budeme pokračovať metódou postupných krokov a dielčich úspechov. To preto, aby sme si boli istí, že tá a tá vec nám už chodí a môžeme pristúpiť k ďalšej súčasti „trojice“. Predpokladám, že MySQL server nám už beží, preto si teraz ako prvé rozbehám http server.

Inštalácia Apache
Inštalačné súbory Apache sa nachádzajú na adrese www.apache.org. Komu sa ich nechce hľadať, može si ich stiahnúť z mojej prepracovanej web stránky s adresou www.mior.host.sk. Tam nájde aj ďalšie zaujímavé súbory. Treba si stiahnúť ten súbor, ktorý je určený pre daný operačný systém, teda v tomto prípade to je súbor ***apache_1_3_11_win32.exe***.
Spustením tohto súboru prebehne pomerne jednoduchá inštalácia, na akú sme pod Windowsami zvyknutí. Defaultne sa Apache na inštaluje do adresára Program Files\Apache Group\Apache.

Základná konfigurácia Apache

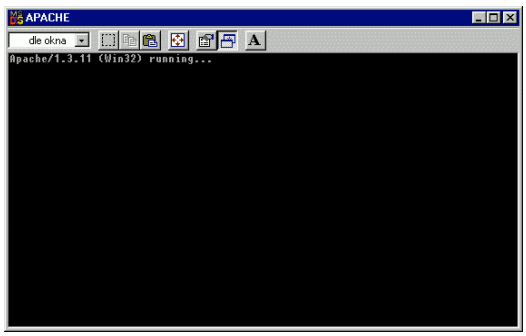
Aby sme apača vôbec po inštalácii spustili, zatiaľ v základnom režime, musíme upraviť hlavný konfiguračný súbor ***httpd.conf***. Nachádza sa v podadresári **Conf** v adresári, kde je inštalovaný Apache. V našom prípade je to **Program Files\Apache Group\Apache\conf**.
V prípade, že tam taký súbor neexistuje, použijeme vzorový súbor ***httpd.conf.default***, ktorý premenujeme na ***httpd.conf***.

Tento konfiguračný súbpr je pomerne zložitý, ale podstatná časť je už predpripravená k použitiu. My len zmeníme niektoré parametre v určitých sekciách.
K editácii *httpd.conf* použijeme ľubovoľný editor, ktorý nezanáša do upravovaných textov žiadne iné formátovacie znaky. Ak používame *Windows Commander*, vnútorný editor pod klávesou F4 plne vyhovuje. Prípadne môžeme použiť *NotePad* z Windows. Editory typu *Word* nepoužívajte!
Teraz si popíšeme úpravu najdôležitejších parametrov:
- vyhľadáme *ServerType* a nastavíme na ***ServerType standalone***
- nájdeme položku *ServerRoot* a napíšeme ***ServerRoot "C:/Program Files/Apache Group/Apache"***
Aj keď to vyzerá ako chyba, naozaj píšeme priame lomky (/) a nie opačné (\), ako sme zvyknutí z DOS-u. Tu vidíte kríženie sveta Unixu a Windows. Ak sme apača inštalovali do iného adresára, zadáme cestu tam.
- skontrolujeme, či máme nastavený ***Port 80***
- ak chceme, aby server posielal hlásenia o problémoch na nejakú e-mailovú adresu, tak zadefinujeme položku *ServerAdmin* na niekto@niekde.kdesi napr. ***ServerAdmin mior@host.sk*** . Toto môžeme aj vynechať, pokiaľ hlásenia nepotrebujeme.
- určíme meno nášho web servera v položke *ServerName*, napr. ***ServerName web.mior.sk*** (zmažeme znak #)
- zadefinujeme adresár, kde sa budú nachádzať naše stránky, napr. ***DocumentRoot "C:/Program Files/Apache Group/Apache/htdocs".*** Podotýkam, že tento adresár musí skutočne existovať v adresárovej štruktúre počítača.
- do položky *DirectoryIndex* doplníme ***DirectoryIndex index.html***
Zatiaľ stačí, súbor ***httpd.conf*** uložíme. V adresári Windows vytvoríme súbor ***hosts***, napr. úpravou a premenovaním súboru ***hosts.sam***, kde doplníme vetu:

192.168.10.152 web.mior.sk

Tým sme nastavili preklad IP adries na doménové mená a naopak. V prípade, že používame náš alebo využívame cudzý DNS server, toto vynecháme.

Ak sme všetko urobili správne, zresetujeme počítač. Po nabehnutí systému z ponuky **Start | Programy | Apache Web Server | Start Apache** server spustíme.
Ak dostaneme DOS okno, ako je na obraázku č. 11-3, server beží správne:



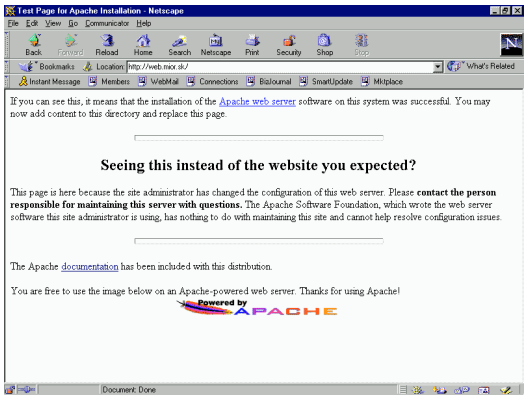
Pozor! Toto okno nikdy nezatvárajte, lebo by ste server zastavili. Môžeme ho však minimalizovaním uložiť na lištu.
V prípade, že okno nenabehlo, alebo sa v ňom objavili podozrivé hlášky, pozrieme sa do súboru ***error.log*** v adresári ***Logs***.

(Upozornenie! V starších verziách Windows 95 bol nevhodný súbor ***winsock.dll***, ktorý spôsoboval, že server Apache nemohol vytvoriť socket na port, alebo hlásil, že nemôže nájsť súbor *winsock32.dll*. Preto si môžete z mojej stránky stiahnuť súbor **W95-winsock2setup.exe**, ktorý tuto chybu odstráni.)

Znakom bezproblémového spustenia http servera je to, že do súboru *error.log* sa nepridala žiadna hláška. Zároveň v tom istom adresári sa objaví súbor ***httpd.pid***, v ktorom je uložené PID číslo servera.

Teraz spustíme ľubovoľného www klienta. Môže to byť *Netscape Navigator*, *Opera* alebo *Internet Explorer*. Zapíšeme lokálnu adresu adresu nášho servera, napr. http://localhost/ alebo priamo meno, ktoré sme zadefinovali v súbore *httpd.conf*, teda v mojom prípade http://web.mior.sk/. Ak sme upravili položku *DirectoryIndex*, tak za poslednú lomku nemusíme zadať *index.html*, jednoducho si to server doplní sám. Ak stránku s týmto názvom nájde v danej adresárovej štruktúre (a to by mal, ak sme pri inštalácii neurobili žiadnu chybu), pošle ju nášmu

klientovi, a my by sme mali uvidieť stránku podobnej obr. č. 11-4:



Ak je to tak, tak ste práve spustili svoj vlastný http server!

Inštalácia PHP

Pristúpime k druhému kroku, a to že nainštalujeme PHP. PHP existuje v dvoch funkčných verziách, a to vo verzii 3.0.x a 4.x.y. My to najprv skúsime s verziou 3.0.x, lebo je stabilnejšia, aspoň pod Winows..
Inštalačné súbory si môžeme stiahnuť z adresy www.php.net, alebo zo zrkadiel www.php.sk prípadne www.php.cz. Samozrejme, aj z mojej adresy, kde nájdete **balík php-3.0.16-win32.zip**.
Tento balík rozzipujeme do adresára, najlepšie **c:\php3**.
Po rozbalení nájdeme v adresári súbor **php3-dist.ini**. Jedná sa o vzor konfiguračného súboru PHP. Tento premenujeme na súbor **php3.ini** a prekopírujeme ho do hlavného adresára Windows, teda **c:\windows**, prípadne **c:\winnt**.
Otvoríme tento súbor v príslušnom editore a vykonáme tieto zmeny:
- v sekcii *Paths and Directories* upravíme riadok
    doc_root = c:\Program Files\Apache group\Apache\htdocs

čím určíme cestu k súborom s našimi stránkami.
- v tej istej sekcii ešte upravíme
    extension_dir = c:\php3

čo je adresár, kde je inštalované PHP.

- v sekcii *Windows Extensions* odkomentujeme (zmažeme znak ;) u riadku
    extension=php3_mysql.dll

čím sme povolili spoluprácu s MySQL.
- takisto odkomentujeme riadok
    extension=php3_odbc.dll

aby sme povolili spoluprácu s ODBC. Čo je ODBC, to si povieme inokedy.

Ostatné parametre tohto konfiguračného súboru necháme tak, ako boli nastavené pri inštalácii.

Všimli sme si, že sme nastavením *extension=php3_mysql.dll* akoby spojili PHP s MySQL. Ale ako povieme Apačovi, že má spolupracovať s PHP?

Konfigurácia Apache na spoluprácu s PHP

Znovu zeditujeme konfiguračný súbor *httpd.conf*. Vyhľadáme položku *ScriptAlias* a na nový riadok dopíšeme
    ScriptAlias /php3/ "c:/php3/" (presne vrátane úvodzoviek).

Takto povieme http serveru, kde sú adresáre pre spracovanie PHP stránok.
Podobne nájdeme položku *AddType application*, kde dopíšeme:

AddType application/x-httpd-php3 .phtml
AddType application/x-httpd-php3 .php3
AddType application/x-httpd-php3 .php

Takto sme uviedli tri MIME typy, takže naše stránky môžu mať príponu ***.phtml***, ***.php3*** alebo ***.php***
K týmto MIME typom musíme nadefinovať konkrétny obslužný program. Stačí, ak dopíšeme:

Action application/x-httpd-php3 "/php3/php.exe"

- do položky *DirectoryIndex* doplníme ***index.php3 index.phtml index.php*** (oddelené medzerami).

(Ak sme inštalovali PHP do iného adresára, musíme cestu k tomuto adresáru patrične upraviť aj v týchto vyššie uvedených položkách obodvoch konfiguračných súborov.)

Takže máme spolu skombinované Apache, PHP a MySQL. Aby sme mohli otestovať vzájomnú súčinnosť, pripravíme si veľmi jednoduchý testovací súbor s menom napr. ***test.php3***, ktorý bude obsahovať iba tri riadky:

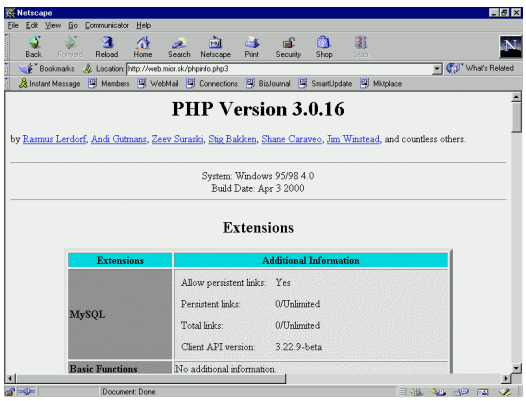
```
<?php
   phpinfo()
?>
```

(Význam jednotlivých položiek bol vysvetlený v PC Revue v seriáli „**Programujeme v PHP**“).

Takto vytvorený súbor uložíme do adresára ***C:\Program Files\Apache Group\Apache\htdocs.***

Po uzavretí všetkých konfiguračných súborov najlepšie zresetujeme počítač. Po novom spustení postupujeme takto:

a) najprv spustíme MySQL. V prípade neistoty, či naozaj funguje, skúsime spojenie pomocou monitora *mysql*
b) spustíme http server Apache už vyššie uvedeným spôsobom. Môžeme prípadne preveriť jeho funkčnosť zavolaním hlavnej stránky ***http://localhost/***
c) do www prezerača zadáme adresu k súboru ***test.php3***, napr. http://localhost/test.php3, alebo plným menom servera http://web.mior.sk/test.php3
d) ak vidíme niečo také ako na obr.11-5:



môžeme smelo zakričať: **„VENCEREMOS!“** (Zvíťazíme!), lebo na testovacej stránke klienta vidíme, že PHP komunikuje aj s MySQL.

A nabudúce pomôžeme tým, čo pracujú na Linuxe.

Miroslav Oravec

Malé veľké databázy II / 2.časť

Ak sme minulú časť seriálu venovali tým, čo budú pracovať s našou „trojicou“ (Apache + PHP + MySQL) na Windowsoch, dnes si prídu na svoje Linuxáci. Dnešná časť je venovaná inštalácii a zprevádzkovaniu apača a „péhápéčka“ na tomto operačnom systéme. Už predom musím upozorniť, že je to pre linuxového začiatočníka pomerne náročná téma, kde inštalácia vo Windows bola proti tomu „sranda“! Problém je v tom, že zatiaľ čo inštalácia vo WinXX bola pomerne jednoznačná, v Linuxe je niekoľko možností. (ale všetky vedú k rovnakému cieľu!) Ale keďže všetky tri produkty boli vyvinuté natívne pre linuxové (un*xové) prostredie, a až potom boli portované na platformu Windows, inštalácia v Linuxe sa nám odmení stabilnosťou a spoľahlivosťou, o ktorej vo Windows môžeme len snívať. Takže - windowsáci si dajú dneskáá na pláži pohov, a linuxáci - ideme na to.

Typy inštalácie

Aj pomerne začiatočník v linuxovom svete, inak prezývaný *newbie* vie, že (skoro) každý program v linuxe je možné inštalovať tromi spôsobmi:

a) z predpripravených inštalačných balíčkov v príslušnom formáte používanej distribúcie, napr. .RPM, .DEB a podobne
b) inštalovaním už skompilovaných súborov - tzv. binárie
c) kompiláciou zdrojových textov

Zatiaľ čo prvé dve možnosti nám pomerne zväzujú ruky, lebo s hotovými balíčkami už nespravíme nič, vlastná kompilácia zdrojákov poskytuje neskonalé možnosti, ako zhotoviť program podľa našich požiadaviek. A to nielen použitím parametrov pri kompilácii, ale aj úpravou samotných zdrojových textov (pokiaľ to licenčná politika k zdrojovým textom dovoľuje - čo býva v Linuxe skoro vždy).
My si ukážeme prvú a poslednú možnosť.

Zdroje súborov

Najprv však konkrétne súbory musíme z niekadiaľ získať. Jednou z možností je stiahnúť ich z Internetu priamo od autorov, teda v našom prípade www.apache.org, www.php.net a www.mysql.com.
Iná možnosť je použiť zdroje programov, ako známe „čerstvé mäso“ - http://freshmeat.net alebo http://slashdot.org. Pre našinca je asi najlepšie využiť cédéčka od ComputerPressu, pretože sa na nich nachádzajú už počeštené programy. To sú také, kde skupina programátorov upravila originálne súbory, aby spoľahlivo zobrazovali aj československé národné znaky. Spravidla sa v názvoch takto upravených súborov nachádza skratka *cs* alebo *cz*. Ale ak sa vám nechce zbytočne hľadať, môžete sa pozrieť na moju stránku www.mior.host.sk, kde nájdete potrebné súbory.

Inštalácia balíkov RPM

Výhodou inštalácie RPM balíkov je, že sa súbory nielen nakopírujú do už prednastavených adresárov, ale sa vytvoria aj potrebné linky a spušťacie skripty. Zároveň sa automaticky program spustí bez potreby resetu počítača. (Tu je veľký rozdiel oproti WinXX - počítač s Linuxom resetujeme len v najhorších prípadoch!)
Budeme potrebovať tieto súbory - *apache-1.x.y.rpm*, *php-3.x.y.rpm*, *php-mysql-3.x.y.rpm*. Vieme, že písmená x a y nahrádzajú čísla verzií. Pre názornosť uvediem príklad mojej inštalácie:
- **apache-1.3.12-4cs.i386.rpm** - obsahuje všetky potrebné súbory http servera
- **apache-manual-1.3.12-4cs.i386.rpm** - obsahuje dokumentáciu k serveru (v angličtine)
- **php-3.0.16-2cs.i386.rpm** - obsahuje súbory PHP preprocesora
- **php-manual-3.0. 16-2cs.i386.rpm** - obsahuje dokumentáciu k PHP (v angličtine)
- **php-mysql-3.0.16-2cs.i386.rpm** - obsahuje potrebné moduly k spolupráci s MySQL

Tak ako všetko v linuxe, aj inštalácie RPM balíkov je možné urobiť rôznymi spôsobmi. Je jedno, ktorý použijeme, všetky vedú k jednému správnemu cieľu. Predpokladajme, že máme MySQL už nainštalované, lebo sme to už urobili dávno a súbory Apache a PHP sme si zkopírovali niekam na disk.
Prvá možnosť je použiť príkazový riadok (symbol # bude značiť príkazový riadok linuxu):

```
# rpm -Uvh apache*.rpm
# rpm -Uhv php*.rpm
```

Druhá možnosť je použiť priamo menežér súborov *Midnight Commander*. Ten veľmi dobre spolupracuje s balíkmi RPM. Kurzor nastavíme na požadovaný balík RPM, stlačíme Enter, čím sa dostaneme k obsahu

balíčka. Presunieme kurzor na súbor s názvom *INSTALL* a znovu stlačíme Enter. Prebehne inštalácia balíka do potrebných adresárov a spustenie programov.

Treťou možnosťou je použitie vhodného RPM menežéra v prostredí X Windows, napr. **GnomeRPM** v prostredí Gnome, alebo **kpackadge** v KDE. Keďže sú tieto prostriedky veľmi intuitívne, nebudem ich tu popisovať.

Inštalácia zo zdrojových textov

Už vieme, že všetky tri programy spolu súvisia a preto je nutné ich vzájomne previazať. Robí sa to hlavne v konfiguračných súboroch, ale pri inštalácii zo zdrojových textov je nutné vykonať určité nastavenia kompilácie. Na to použijeme parametre v príkazovom riadku. Preto budeme preskakovať z kompilácie Apache do kompilácie PHP a naspäť, aby sme zabezpečili požadované previazanie. Samozrejme, po kompilácii sa ešte nevyhneme nastaveniu v konfiguračných súboroch.
Zdrojové texty súborov sa spravidla nachádzajú zkomprimované vo formáte ***.tar.gz***. Predpokladajme, že vlastníme zdrojové texty v súboroch:
- **apache_1.3.12.tar.gz** - čo je balíček súborov http servera, ktorý skopírujeme napr. do adresára */opt/apache* (tento adresár si samozrejme môžeme ľubovoľne zvoliť, je to len pomocný adresár)
- **php_3.0.16.tar.gz** - čo je balíček so súbormi PHP preprocesora, ktorý skopírujeme do adresára */opt/php*.
Poďme spolu postupnými krokmi k cieľu:

1) V obidvoch uvedených adresároch „ztarované“ balíky rozbalíme:

   # gzip -d *.gz
   # tar xf *.tar

2) V adresári */opt/apache* pristúpime ku konfigurácii prekladu Apache, kde parametrom *prefix* stanovíme, kam chceme http server nainštalovať. Pre náš príklad to môže byť do adresára */opt/www*:

   # ./configure --prefix=/opt/www

3) Prejdeme do adresára */opt/php* a zkonfigurujeme preklad PHP. Keďže chceme databázovú podporu pre MySQL a zároveň spoluprácu s Apačom, toto uvedieme v príslušných parametroch:

   # ./configure --with-mysql --with-apache_1.3.12 --enable-track-vars

   [Alternatívou k príkazu **./configure** je spustenie skriptu **./setup**, ktorý sa spýta na niekoľko otázok a na základe odpovedí potom spustí príkaz **./configure** s príslušnými parametrami.]

4) Po úspešnej konfigurácii spustíme kompiláciu PHP v adresári */opt/php*:

   # make
   # make install

5) Ak prebehla kompilácia PHP v poriadku, presunieme sa opäť do adresára */opt/apache*, kde dokončíme konfiguráciu kompilácie Apache pre podporu s modulom PHP:

   # ./configure --prefix=/opt/www --activate-module=src/modules/php3/libphp3.a

6) V adresári */opt/php* prekopírujeme:

   # cp php3.ini-dist /usr/local/lib/php3.ini

   čím vytvoríme súbor *php3.ini.*

7) Pristúpime ku kompilácii Apača:

   # make
   # make install

V adresári */opt/www* sa vytvorí adresárová štruktúru */bin, /sbin, /htdocs, /log*. Nová binárna verzia démonu ***httpd*** sa nachádza v adresári */sbin* (inštalácia pomocou RPM balíkov môže vytvoriť inú adresárovú štruktúru). Touto novou verziou nahradíme starú verziu *httpd*.

Zabezpečenie spúšťania démona httpd

Aby sme zabezpečili automatické spúšťanie démona ***httpd*** vždy po štarte počítača, vykonáme nasledujúce kroky, ktoré sú platné pre Red Hat Linux:

Vytvoríme štartovací skript s názvon **httpd**. Jeho vzor je na výpise č. 12-1:

```
#!/bin/sh

# Source function library.
. /etc/rc.d/init.d/functions

# See how we were called.
case "$1" in
  start)
      echo -n "Starting httpd: "
      daemon httpd
      echo
      touch /var/lock/subsys/httpd
      ;;
  stop)
      echo -n "Shutting down http: "
      killproc httpd
      echo
      rm -f /var/lock/subsys/httpd
      rm -f /var/run/httpd.pid
      ;;
  status)
      status httpd
      ;;
  restart)
      $0 stop
      $0 start
      ;;
  reload)
      echo -n "Reloading httpd: "
      killproc httpd -HUP
      echo
      ;;
  *)
      echo "Usage: $0 {start|stop|restart|reload|status}"
      exit 1
esac

exit 0
```

Tento skript uložíme do adresára */etc/rc.d/init.d*.

Do adresára */etc/rc3.d* urobíme symbolickú linku s číslom niekde na konci číselnej rady, napr. ***S98apache***, ktorá bude ukazovať na súbor */etc/rc.d/init.d/httpd*.
Ako je z výpisu vidieť, je možné činnosť httpd démona ovládať príkazmi:

 # /etc/rc.d/init.d/httpd start
alebo
 # /etc/rc.d/init.d/httpd stop

alebo *restart*, *reload* a *status*.

Ak sme všetko vykonali správne, môžeme pristúpiť ku konfigurácii Apache, ktorá je veľmi obdobná, ako v prostredí Windows.

Konfigurácia Apache

Popíšeme si jednotlivé úkony. Editujeme dobre známy súbor *httpd.conf*:
- vyhľadáme *ServerType* a nastavíme na ***ServerType standalone***
- určíme meno nášho web servera v položke *ServerName*, napr. ***ServerName web.doma.sk*** (zmažeme znak #)
- nájdeme položku *ServerRoot* a napíšeme ***ServerRoot "/etc/httpd"***
- skontrolujeme, či máme nastavený ***Port 80***
- ak chceme, aby server posielal hlásenia o problémoch na nejakú e-mailovú adresu, tak zadefinujeme položku *ServerAdmin* na niekto@niekde.kdesi napr. ***ServerAdmin mior@host.sk*** . Toto môžeme aj vynechať, pokiaľ hlásenia nepotrebujeme.
- odpoznámkujeme (zmažeme znak #):

LoadModule php3_module /usr/lib/apache/libphp3.so
AddModule mod_php3.c
EncodeContentType application/x-httpd-php
EncodeContentType application/x-httpd-php3
EncodeContentType application/x-httpd-phps

aby server Apache natiahol potrebné moduly pre spoluprácu s PHP.
- nastavíme, aby Apache správne obsluhoval stránky PHP. Nájdeme položku *AddType application*, kde dopíšeme:

AddType application/x-httpd-php3 .phtml
AddType application/x-httpd-php3 .php3
AddType application/x-httpd-php3 .php

Takto sme uviedli tri MIME typy, takže naše stránky môžu mať príponu ***.phtml***, ***.php3*** alebo ***.php***
- nastavíme, kde chceme mať uložené naše stránky, napr. ***DocumentRoot "/home/httpd/html"***
- do položky *DirectoryIndex* doplníme ***DirectoryIndex index.html index.php3 index.php***

Zatiaľ stačí, súbor ***httpd.conf*** uložíme. V adresári */etc* vytvoríme súbor ***hosts***, kde doplníme vetu:

192.168.10.153 web.doma.sk

Tým sme nastavili preklad IP adries na doménové mená a naopak. V prípade, že používame náš alebo využívame cudzý DNS server, toto vynecháme.

Konfigurácia PHP

Pri konfigurácii PHP stačí editovať súbor *php3.ini*, kde odpoznámkujeme položku:

extension=mysql.so

Tým sme stanovili, že v prípade potreby bude preprocesor kooperovať s MySQL serverom.

Test spojenia

Takže máme spolu skombinované Apache, PHP a MySQL. Aby sme mohli otestovať vzájomnú súčinnosť, pripravíme si veľmi jednoduchý testovací súbor s menom napr. ***test.php3***, ktorý bude obsahovať iba tri riadky:

```
<?php
   phpinfo()
?>
```

Takto vytvorený súbor uložíme do adresára, kam je nastavená položka *DocumentRoot*, teda */home/httpd/html*.

Uložíme všetky otvorené súbory a vykonáme tieto kroky:

e) najprv overíme, či máme spustené MySQL. Použijeme príkaz:

 # ps ax¦grep mysql

Ak dostaneme niečo podobné, ako na výpise 12-2, mysql server beží:

```
  577 ?        S      0:00 sh /usr/bin/safe_mysqld --user=mysql --datadir=/var/l
  609 ?        S      0:00 /usr/sbin/mysqld --basedir=/ --datadir=/var/lib/mysql
  622 ?        S      0:00 /usr/sbin/mysqld --basedir=/ --datadir=/var/lib/mysql
  623 ?        S      0:00 /usr/sbin/mysqld --basedir=/ --datadir=/var/lib/mysql
  851 pts/0    S      0:00 grep mysql
```

V prípade neistoty, či naozaj funguje, skúsime spojenie pomocou monitora *mysql*

Ak mysql server nebeží, spustíme ho príkazom:

 # mysqld &

a znova overíme jeho beh.

f) podobne príkazom:

 # ps ax¦grep http

overíme beh http démona. Mali by sme dostať niečo ako na výpise 12-3:

```
  572 ?        S      0:00 httpd
  584 ?        S      0:00 httpd
  585 ?        S      0:00 httpd
  586 ?        S      0:00 httpd
  587 ?        S      0:00 httpd
  588 ?        S      0:00 httpd
  589 ?        S      0:00 httpd
  590 ?        S      0:00 httpd
  591 ?        S      0:00 httpd
  872 pts/0    S      0:00 grep httpd
```

Tiež môžeme použiť príkaz:

 # /etc/rc.d/init.d/httpd status

ktorý vypíše status démona httpd - výpis 12-4:

```
httpd (pid 591 590 589 588 587 586 585 584 572) is running...
```

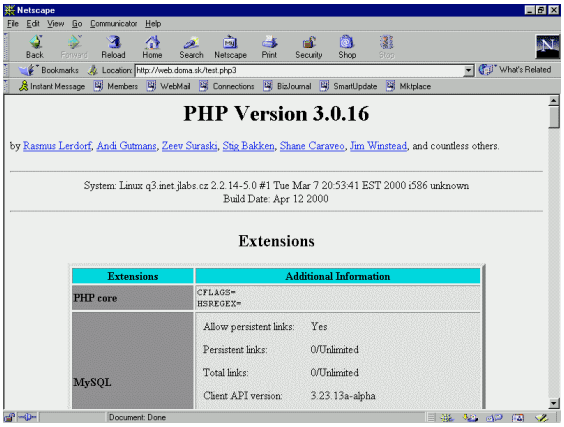
Ak server httpd nebeží, spustíme ho príkazom:

 # /etc/rc.d/init.d/httpd start

a overíme jeho beh.

Môžeme prípadne preveriť jeho funkčnosť zavolaním hlavnej stránky http://localhost/**,** prípadne http://web.doma.sk/ alebo IP adresou http://192.168.10.153 v niektorom z obľúbených prezeračov. Dostaneme úvodnú obrazovku, ako v minulej časti, alebo ak sme inštalovali počeštenú verziu, dostaneme výsledok, ako je na obr.12-5:

g) do www prezerača zadáme adresu k súboru ***test.php3***, napr. http://localhost/test.php3, alebo plným menom servera http://web.doma.sk/test.php3. Ak vidíme niečo také ako na obr.11-5:



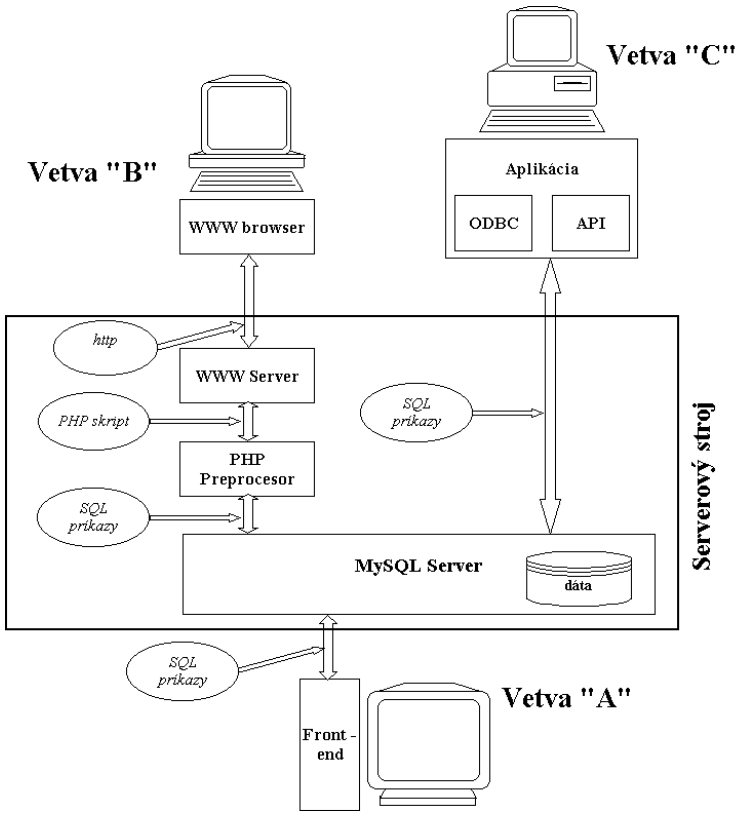
vieme, že funguje Apache s PHP vrátane MySQL.

A to bolo cieľom nášho snaženia. V tomto bode sa cesty priaznivcov Windows a milovníkov Linuxu opäť stretávajú. Tak si ešte poležme niekde na pláži, poobzerajme pekné baby (dámy zase chalanov) a nazbierajme sily na tvorbu prvej oknoidnej databázovej aplikácie. Ak môžete, pozrite si predbežne niečo o tvorbe HTML stránok (ale nemusíte).

Malé veľké databázy II / 3.časť

Leto je pravdepodobne za nami a začal nový školský rok. Aj ja vás vítam v našej škole databáz a keďže sme oddýchnutí, hneď sa pustíme do poriadnej práce. Dúfam, že sa inštalácie podarili a máme všetci všetko správne rozbehané. Dnes si ukážeme, ako „serverová trojka“ Apache + PHP + MySQL funguje v praxi.

Trocha teórie
My už vieme, že naša „serverová trojka“ beží spravidla na jednom počítači. Nie je to však nutnosťou. Je kľudne možné vo zvláštnych prípadoch mať na jednom počítači Apache + PHP a na druhom zase MySQL. Mám to prakticky overené, ale zatiaľ som to v praxi nepotreboval. V takom prípade sa pozmenia trochu konfiguračné súbory a samozrejme aj PHP skripty.
Pre jednoduchosť budeme uvažovať, že „trojka“ beží na jednom počítači a klientska aplikácia na inom počítači. Je jedno, na akom operačnom systéme pobežíme, keďže sa jedná o rovnaké princípy. Na prípadné odchylky vás upozorním.
Nesmieme zabudnúť, že najdôležitejšie sú pre nás naše dáta. Jadrom celého systému naďalej zostáva MySQL server. To ostatné, Apache a PHP, je len technologické okolie, slúžiace k tomu, aby sme sa k našim dátam dostali. Na obr. č. 13-1 sú znázornené rôzne možné spôsoby, ako sa dá k dátam pristúpiť:



Vetva „A“ popisuje najjednoduchší spôsob - priamy prístup pomocou tzv. front-end aplikácie od výrobcu SQL serveru. Takou front-end aplikáciou je aj nám dobre známy monitor ***mysql***. Touto cestou sme pracovali celé predchádzajúce lekcie a vieme, že je vhodná iba pre databázových administrátorov.

Vetva „B“ popisuje prístup k datám pomocou http servera. Užívateľ sedí za http klientom, čo môže byť ľubovoľný internetovský prehliadač - tzv. ***browser***, napr. Netscape Navigator, Opera, Internet Explorer, Mozilla a iné. Pomocou browsera posiela konkrétne požiadavky na dáta, tie http server príjme, odovzdá PHP preprocesoru. Ten sa spojí s SQL serverom, získa požadované dáta, upraví do vhodnej podoby a cestou http servera sa zobrazia na obrazovke klienta. Toto je veľmi vhodná cesta, lebo prístup k jednotlivým dátam je možný z ľubovoľného miesta na svete, kde je prijateľné spojenie a dostupný browser. Týmto spôsobom sa budeme zaoberať niekoľko lekcií.

Vetva „C“ znázorňuje prístup k dátam, keď na klientskom počítači beží určitá natívna aplikácia. Tá môže byť napísaná vo vyššom programovacom jazyku, napr. Pascal, Delphi, C, Visual Basic a iné, ktoré umožňujú používať SQL príkazy. Ba dokonca to môže byť aj Excel, do ktorého si pritiahneme požadované dáta z SQL servera. Klientská aplikácia komunikuje s SQL serverom buď priamo, pomocou API funkcií daného SQL servera, alebo nepriamo, pomocou ODBC. Táto varianta prístupu k dátam sa využíva pri tvorbe rôznych účtovných programov, kde sa riešia špecializované funkcie programu, ako tlač do formulárov a pod., ktoré nie je možné zabezpečiť cestou vetvy „B“. Aj my sa budeme trochu touto vetvou zaoberať a ukážeme si niekoľko príkladov.

Vráťme sa teraz k vetve „B“. Aby sme mohli písať PHP skripty, musíme ovládať syntaxiu príkazov. Keďže v PC Revue už bežal seriál o programovaní v PHP, nebudeme sa dopodrobna zaoberať jednotlivými príkazmi. My sa pozrieme podrobne na tie príkazy, ktoré spolupracujú s MySQL serverom.

PHP funkcie
PHP má bohatú škálu funkcií na komunikáciu s MySQL serverom. Je ich dokopy 33, avšak na základnú prácu nám postačí iba niekoľko z nich. Ich zoznam je uvedený v tabuľke č.13-2:

Tabuľka č.13-2: Najzákladné funkcie PHP pre spojenie s MySQL serverom

Názov	Zápis	Popis
MySQL_Connect	MySQL_Connect(<*počítač*>, <*užívateľ*>, <*heslo*>)	Vytvorí spojenie s databázou.
MySQL_Close	MySQL_Close(<*spojenie*>)	Uzavrie spojenie s databázou
MySQL_Select_DB	MySQL_Select_DB(<*meno_db*>, <*spojenie*>)	Vytvorí spojenie s databázou.
MySQL_Query	MySQL_Query(<sql-príkaz>, <spojenie>)	Vykoná SQL príkaz.
MySQL_Num_Rows	MySQL_Num_Rows(<*výsledok*>)	Zistí počet záznamov výsledku
MySQL_Fetch_Array	MySQL_Fetch_Array(<*výsledok*>)	Načíta jeden riadok výsledku do asociatívneho poľa.
MySQL_Result	MySQL_Result(<*výsledok*>, <*záznam*>, <*položka*>)	Získanie jednej položky výsledku dopytu
MySQL_Free_Result	MySQL_Free_Result(<*výsledok*>)	Uvoľnenie výsledku z pamäte
MySQL_Affected_Rows	MySQL_Affected_Rows(<*spojenie*>)	Vracia počet záznamov, ovplyvnených príkazom INSERT,UPDATE a DELETE
MySQL_Create_DB	MySQL_Create_DB(<*meno_db*>, <*spojenie*>)	Vytvorí novú databázu
MySQL_Drop_DB	MySQL_Drop_DB(<*meno_db*>, <*spojenie*>)	Zmaže databázu

V PHP je dobrým zvykom funkcie na spoluprácu s databázami pomenovávať podľa databázového stroja. Preto aj funkcie k MySQL začínajú slovom MySQL_. Za ním nasleduje meno funkcie.

Náš prvý skript
Je načase, aby sme pristúpili k tvorbe nášho prvého skriptu PHP. Jeho úloha bude veľmi jednoduchá. Po svojom zavolaní spočíta počet záznamov v tabuľke **kniha** v databáze **KNIZNICA**, a tento počet vypíše na obrazovku browsera.
Vieme, že skripty PHP sa zapisujú medzi znaky <**?** a **?**>. Ako prvé musíme nadviazať spojenie s MySQL serverom. Na to slúži funkcia ***MySQL_Connect***. Formálny zápis vyzerá takto:

$spojenie = MySQL_Connect(<počítač>, <užívateľ>, <heslo_užívateľa>);

Parametrami funkcie sú:
- <***počítač***> - názov počítača, kde beží MySQL server, v tomto prípade “***localhost***“, pretože apač spolu s PHP bežia na tom istom serveri ako MySQL
- <***užívateľ***> - meno užívateľa, ktorý sa do databáze pripája, napr. “***root***“
- <***heslo_užívateľa***> - heslo užívateľa na prístup k databázam, v mojom prípade “***heslo***“

Poznámka: Tento príklad je ilustračný a nerieši bezpečnosť siete. Preto si pre pochopenie dovolíme používať roota a jeho heslo.

Táto funkcia vracia číslo spojenia, pod ktorým beží. S týmto číslom, ktoré je uložené do premennej ***$spojenie*** (názov premennej si môžeme zvoliť), potom budeme operovať v ďalších príkazoch. Takže zápis na spojenie k nášnu SQL serveru bude:

$spojenie = MySQL_Connect(“localhost”, “root”, “heslo”);

Vieme, že každý PHP príkaz musí byť oddelený stredníkom - bodkočiarkou (**;**).

Ak sa spojenie nepodarí, funkcia vratí hodnotu *false*. Neskôr si ukážeme, ako sa dá takéto chybové hlásenie ošetriť v náš prospech.
Keď už máme úspešne spojenie nadviazané, musíme určiť, do ktorej databáze chceme pristupovať. To dosiahneme funkciou ***MySQL_Select_DB***. Ako sám názov naznačuje, funkcia vyselektujezadanú databázu za aktívnu. Formálny zápis je:

MySQL_Select_DB(<názov_databáze>, <spojenie>);

Názov_databáze je jasný, v našom prípade **KNIZNICA**. *Spojenie* je číslo spojenia, ktoré vráti funkcia ***MySQL_Connect***. V prípade, že uvažujeme iba s jedným spojením, môžeme tento parameter vynechať. Ak sme si nie istí, či môže nastať viac spojení, je dobré tento parameter použiť.
V našom prípade to bude:

MySQL_Select_DB(“kniznica”, $spojenie);

Následuje zadanie SQL príkazu. Na to slúži funkcia ***MySQL_Query***. Formálny zápis je:

$vysledok = MySQL_Query(<SQL_príkaz>, <spojenie>);

Táto funkcia vykoná na danom spojení zadaný SQL príkaz nad práve aktívnou databázou a vráti identifikátor výsledku. Ak pri vykonávaní SQL príkazu dôjde k chybe, vráti funkcia hodnotu *false*. Ak nepoužijeme parameter <*spojenie*> neuvedieme, použije sa posledné vytvorené spojenie. My chceme vykonať príkaz selekcie nad tabuľkou **kniha**:

$vysledok = MySQL_Query(“select * from kniha”, $spojenie);

Pozor na to, že MySQL server je *case sensitive* pri názvoch tabuliek, teda nie je jedno, či napíšeme **kniha** alebo **KNIHA**!

A keďže chceme zistiť počet záznamov v tabuľke **kniha**, použijeme funkciu:

$pocet = MySQL_Num_Rows(<výsledok>);

Funkcia vráti do premennej ***$pocet*** počet záznamov, ktoré obsahuje **výsledok**. Pri chybe vráti hodnotu *false*. Náš prípad teda bude:

$pocet = MySQL_Num_Rows($výsledok);

Keď už máme všetko spočítané, chceme naše snaženie oznámiť žiadateľovi. Použijeme funkciu **echo** takto:

echo “V tabulke KNIHA je $pocet zaznamov”;

echo zabezpečí výpis reťazca, uvedeného v úvodzovkách. Ak je súčasťou reťazca aj premenná, vypíše sa jej obsah.

Poznámka:
Ak by sme reťazec uzavreli do apostrofov a nie do úvodzoviek, tak sa vypíše reťazec tak, ako je napísaný, teda nie obsah premennej, ale jej meno.

Na záver každej operácie je slušnosťou spojenie uzavrieť. To dosiahneme funkciou:

MySQL_Close(<spojenie>);

V našom prípade:

MySQL_Close($spojenie);

Ako teda bude vyzerať náš prvý skript? Pomenujeme ho ***priklad1.php*** a jeho výpis je na výpise č.13-3:

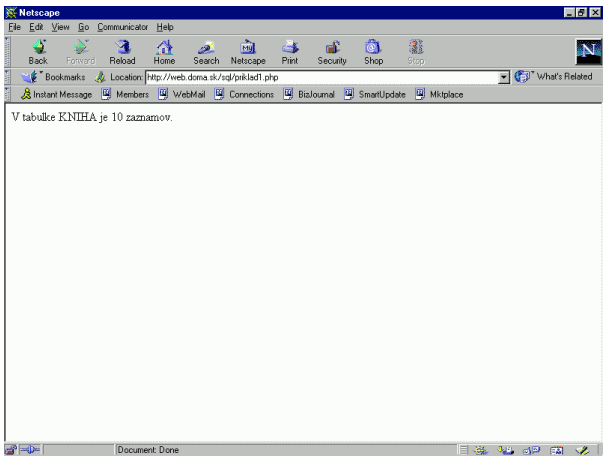
```
<?
      $spojenie = MySQL_Connect("localhost","root","heslo");
      MySQL_Select_Db("kniznica",$spojenie);
      $vysledok = MySQL_Query("select * from kniha", $spojenie);
      $pocet = MySQL_Num_Rows($vysledok);
      echo "V tabulke KNIHA je $pocet zaznamov.";
      MySQL_Close($spojenie);
?>
```

Skript treba napísať vo vhodnom editore, vytvárajúcom klasický ASCII text, teda nie vo Worde a podobne. V takom prípade by sa mohlo stať, že PHP preprocesor by nevedel správne interpretovať znaky, ktoré v súbore zanechal nevhodný editor, ale my ich na obrazovke nevidíme. Azda najvhodnejším editorom je *Programer's File Editor*. Umožňuje napísané súbory previesť aj do unix formy, vytlačiť s číslovaním riadkov a podobne. Stiahnúť si ho môžete z mojej www stránky www.mior.host.sk.

Keď máme skript napísaný, uložíme ho na náš http server. Môžeme ho uložiť priamo do adresára, uvedeného v *DocumentRoot* v súbore *httpd.conf*. Je vhodné vytvoriť nový adresár, napr. ***sql***, a do neho uložiť naše cvičné skripty. Spustíme vhodný prehliadač, zadáme adresu web servera, cestu ku skriptu a názov skriptu, napr:

http://web.doma.sk/sql/priklad1.php

Ak dostaneme niečo podobné, ako na obr.č.13-4, vidíme, že náš prvý skript v spojení s databázou funguje:



Náš druhý skript

No, aký je počet záznamov v tabuľke **kniha** už vieme, ale my by sme chceli vypísať aj obsah tabuľky. Preto si teraz vytvoríme druhý skript, ktory pomenujeme ***priklad2.php***. Po ňom budeme chcieť to, čo po prvom skripte, ale naviac chceme, aby sa vypísal aj obsah stĺpcov **nazov** a **autor** z tabuľky **kniha**.

Už vieme, že výsledok SQL dopytu je uložený v premennej ***$vysledok***. Na vyzískanie potrebných údajov môžeme použiť viac funkcií, ale asi najvhodnejšia bude funkcia ***MySQL_Fetch_Array()***, ktorá prečíta jeden

záznam z premennej ***$vysledok*** a obsah jeho položiek uloží do asociatívneho poľa, ktoré si môžeme pomenovať ***$zaznam***:

```
$zaznam = MySQL_Fetch_Array($vysledok);
```

Obsah každej položky záznamu je uložený do prvku, ktorý má názov rovnaký ako je názov položky v tabuľke. To veľmi uľahčí prácu so záznamom. Potom sa funkcia posunie na ďalší záznam. Ak funkcia vráti všetky záznamy, vráti hodnotu *false*. Túto vlastnosť môžeme výhodne využiť v cykle načítavania takto:

```
while ($zaznam = MySQL_Fetch_Array($vysledok))
        echo $zaznam[“nazov”].” -- “.$zaznam[“autor”].”<BR>\n”;
```

Čo je to za zvláštny zápis **$zaznam[“nazov”]** ? To je práve získanie obsahu premennej ***$zaznam***, ktorá zodpovedá stĺpcu s menom **nazov** v príslušnej tabuľke **kniha**. Obdobne získame obsah položky **autor**, a pomocou ***echo*** to vypíšeme na obrazovku. Aby bol text lepšie čitateľný, vložili sme medzi obe polia dve čiarky s medzerami a na konci sme dali znak tvrdého zalomenia <***BR***> s novým riadkom ***\n***. Predpokladám, že spájanie reťazcov nám v PHP nerobí problémy, ale pre zopakovanie pripomeniem, že sa to jednoducho robí pomocou bodky.

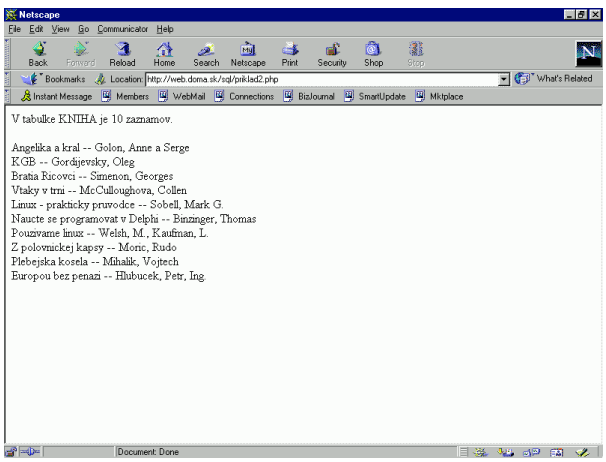
Ako teda bude vyzarať zápis skriptu ***priklad2.php***? Tak ako na výpise č.13-5:

```
<?
      $spojenie = MySQL_Connect("localhost","root","heslo");
      MySQL_Select_Db("kniznica",$spojenie);
      $vysledok = MySQL_Query("select * from kniha", $spojenie);
      $pocet = MySQL_Num_Rows($vysledok);
      echo "V tabulke KNIHA je $pocet zaznamov.";
      echo "<P>";
      while ($zaznam = MySQL_Fetch_Array($vysledok))
            echo $zaznam["nazov"]." -- ".$zaznam["autor"]."<BR>\n";
      MySQL_Close($spojenie);
?>
```

Ak vás zaujal zápis echo ***“<P>”***, tak vedzte, že sa jedná o vytvorenie nového odstavca.
Skript uložíme, prekopírujeme na príslušné miesto vo web serveri a v prehliadači ho zavoláme takto:

http://web.doma.sk/sql/priklad2.php

Prehliadač sa spojí so serverom, ten vykoná príslušnú prácu a na obrazovke sa zobrazí obr. č.13-6:



Už viackrát sme si povedali, že http server síce príjme požiadavku na stránku s príkazmi PHP, ale jej vytvorenie nechá práve na preprocesor PHP. Ten, len čo spracuje úlohy spojené s databázou, vytvorí súbor vo formáte

jazyka HTML, a ten cestou servera odošle žiadateľovi. Na výpise č.13-7 je zdrojový text takej HTML stránky, ktorú vytvorí PHP po spracovaní skriptu ***priklad2.php***:

```
V tabulke KNIHA je 10 zaznamov.<P><BR>Angelika a kral -- Golon, Anne a
Serge<BR>
KGB -- Gordijevsky, Oleg<BR>
Bratia Ricovci -- Simenon, Georges<BR>
Vtaky v trni -- McCulloughova, Collen<BR>
Linux - prakticky pruvodce -- Sobell, Mark G.<BR>
Naucte se programovat v Delphi -- Binzinger, Thomas<BR>
Pouzivame linux -- Welsh, M., Kaufman, L.<BR>
Z polovnickej kapsy -- Moric, Rudo<BR>
Plebejska kosela -- Mihalik, Vojtech<BR>
Europou bez penazi -- Hlubucek, Petr, Ing.<BR>
```

Toto dosiahneme, keď po prijatí odpovede, aká je na obr.č.13-6, prejdeme do menu **View - Page Source** (v NetScape 4.7). Môžeme porovnať, že výpis na obr. 13-5 nie je vôbec zhodný s týmto HTML výpisom, ale je jeho výsledkom. A to je práve úlohou PHP.

Skúsení tvorcovia HTML stránok namietnu, že tieto skripty sú síce funkčné, ale nečisté. Odporujú zásadam pre tvorbu HTML stránok a niektoré prehliadače by ich nemuseli interpretovať správne. Preto skript ***priklad2.php*** „oblečieme“ do správneho HTML kódu. Zároveň ošetríme chybové hlásenia. Vieme, že ak nastane chyba pri behu skriptu, chybové hlásenie sa zobrazí na obrazovke a beh skriptu sa ukončí. Omnoho prijateľnejšie a užívateľsky príjemnejšie je vlastné ošetrenie chýb. Pomocou znaku **@** potlačíme zobrazovanie štandardných chybových hlásení a vytvoríme vlastné chybové hlásenia. Je to veľmi jednoduché. Skoro každá funkcia v PHP pri chybovom stave vráti hodnotu *false*. Takže také ošetrenie pripojenia k MySQL bude vyzerať asi takto:

```
if (!$spojenie):
        echo "Nepodarilo sa pripojit k MySQL.<BR>\n";
        break;
endif;
```

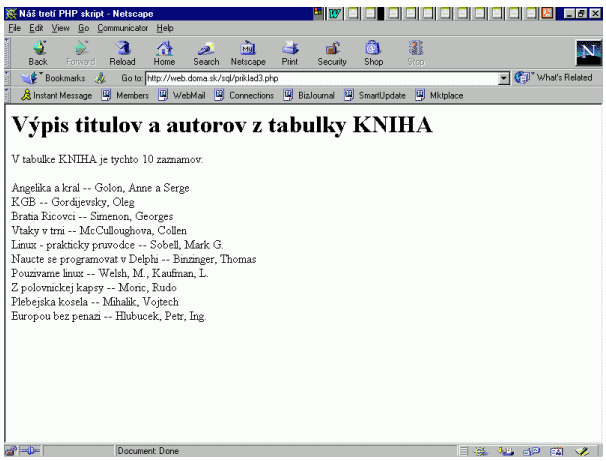
čo môžeme interpretovať - ak nie je spojenie, vypíš čosi s zastav sa!
Efektné, však?

Správne a podľa všetkých regulí vytvorevý skript ***priklad3.php***, v ktorom budú ošetrené aj chyby, je na výpise č.13-8:

```
<HTML>
•
  <HEAD>
•
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;windows-1250">
    <TITLE>Náš tretí PHP skript</TITLE>
  </HEAD>
<BODY>
<H1>Výpis titulov a autorov z tabulky KNIHA</H1>
<?
do {
   @$spojenie = MySQL_Connect("localhost","root","heslo");
   if (!$spojenie):
      echo "Nepodarilo sa pripojit k MySQL.<BR>\n";
      break;
   endif;
   MySQL_Select_DB("kniznica",$spojenie);
   @$vysledok = MySQL_Query("select * from kniha", $spojenie);
   if (!$vysledok):
      echo "Došlo k chybe spracovania SQL príkazu.<BR>\n";
      break;
```

```
   endif;
   $pocet = MySQL_Num_Rows($vysledok);
   echo "V tabulke KNIHA je tychto $pocet zaznamov:<BR>\n";
   echo "<P>";
   while ($zaznam = MySQL_Fetch_Array($vysledok))
         echo $zaznam["nazov"]." -- ".$zaznam["autor"]."<BR>\n";
   MySQL_Close($spojenie);
   } while (false);
?>

</BODY>
</HTML>
```

Znova ho uložíme na web a zavoláme v prehliadači, kde uvidíme obr.č.13-9:



Aspoň trochu user friendly program

Tvrdíte, že to stále nie je to pravé orechové? Áno, súhlasím, veď sme iba na začiatku. Pôvodne som chcel už ukončiť dnešnú lekciu, ale mám pre vás takú perličku. Na záver si vytvoríme aspoň trochu user-friendly aplikáciu do našej knižnice. Bude ilustrovať, čo sme sa doteraz naučili a bude využívať aj prístupové práva, ktoré sme tak pracne tvorili. Jej činnosť bude táto - po zavolaní príslušnej stránky sa zobrazí vstupný formulár. Po jeho vyplnení sa návštevníkovi (alebo zamestnancovi) knižnice zobrazí zoznam kníh v knižnici. Tak ideme na to!

Najprv si musíme vytvoriť súbor ***aplikacia1.htm***. Jeho obsah je na výpise č.13-10:

```
•
<HTML>
•
<HEAD>
•
    <META NAME="Generator" CONTENT="Golden HTML Editor 4.8.1">
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1250">
    <TITLE>Náš komunikatívny program</TITLE>
</HEAD>
<BODY>
<H1>Zadaj prístupové meno a heslo do databáze:</H1>
<PRE>
<FORM ACTION="priklad4a.php" METHOD=POST>
```

```
  Meno      : <INPUT NAME=meno><BR>
  Heslo     : <INPUT NAME=heslo><BR>
<INPUT TYPE=Submit VALUE="Odoslanie  formulara">
</FORM>
</PRE>
</BODY>
</HTML>
```

Takto sme vytvorili dopytovací formulár, kde zadáme meno a heslo na prístup do databáze. Ako je zrejmé, tieto sa uložia do premenných ***meno*** a ***heslo***. Vyplnené údaje nakoniec odošleme tlačítkom s nápisom „*Odoslanie formulára*“. Všimnime si riadok

<FORM ACTION="priklad4a.php" METHOD=POST>

ktorý hovorí, ktorému skriptu budú zadané údaje odoslané k spracovaniu. Je to skript ***priklad4a.php***, ktorý je čiastočne upravený náš skript ***priklad4.php*** z výpisu č.13-8. Zmena spočíva v úprave riadku

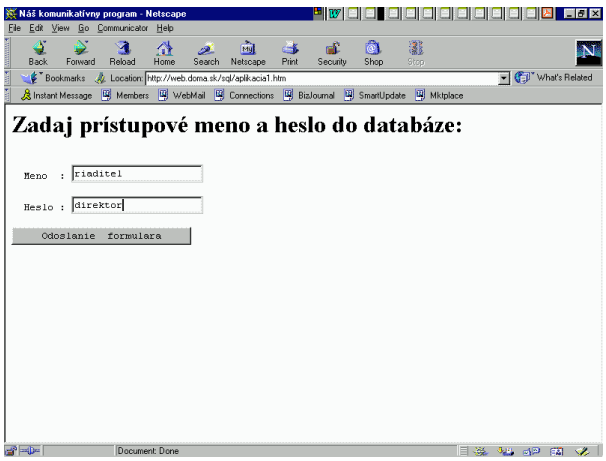
@$spojenie = MySQL_Connect("localhost",$meno,$heslo);

kde sme „napevno“ napísané meno užívateľa a jeho heslo - teda “*root*” a “*heslo*” vymenili za premenné z formulára. Takto je nový skript variabilný, lebo za premenné môžeme dosadiť ľubovoľné meno a heslo. Za riadok so spojením môžeme doplniť ešte jeden riadok:

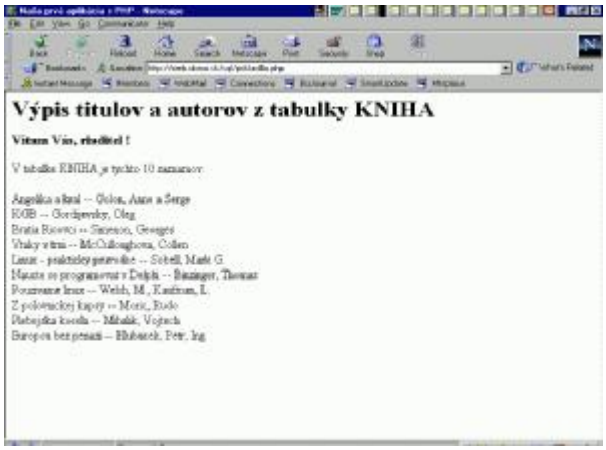
echo "<H3>Vítam Vás, $meno !</H3>"."<P>";

ktorý zabezpečí privítacie hlásenie podľa mena prihlasujúceho.

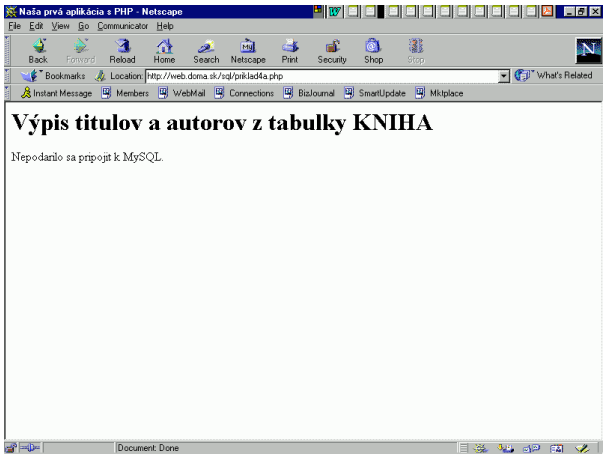
A ako taký formulár vyzerá? Tak ako na obrázku č.13-11:



Vidíme, sme vyplnili prihlasovacie meno ***riaditel*** a jeho heslo ***direktor***. Stlačíme šedé tlačítko a ak sú zadané údaje správne, dostaneme okno, ako je na obr. č. 13-12:



Ak nie, nastane chyba. Keďže sme chyby ošetrili a nahradili našimi chybovými hláškami, môžeme v prípade zle zadaného mena alebo hesla uvidieť takýto obrázok (obr.č.13-13):



Dobrí fajnšmekri by vedeli do skriptu vložiť zopár formátovacích tagov, ktoré by urobili peknú tabuľku okolo výpisu a podobne. To však nie je cieľom tejto lekcie.
Teraz máme hotovú prvú aplikáciu. Nie je ešte dokonalá, ale je aspoň funkčná. Vyskúšajte si zadávať všetky mená a heslá, ktoré sme definovali v 9. a 10. časti seriálu. Aspoň zistíte, ako reaguje.
Všetky zdrojové texty nemusíte pracne opisovať, ale si ich môžete stiahnúť z mojej web stránky na adrese www.mior.host.sk. Zároveň vás tam čaká niekoľko zaujímavých noviniek.

Malé veľké databázy II. /4.časť - intermezzo

Namiesto predslovu
Už je to tu! Aj vy isto poznáte ten prípad zo školských lavíc. V našom obľúbenom predmete sme sa ledva prehrýzli povinnou úvodnou teóriou a prešli sme konečne k praxi, ktorá nás toľko zaujíma. A jedného dňa, namiesto praktickej interesantnej hodiny príde učiteľ a zasype nás množstvom daľšej nezáživnej teórie. Vtedy som si myslel, že nám to hádam robí naschvál. Ale on dobre vedel, že sme sa dostali do štádia, keď na ďaľšiu praktickú činnosť naše teoretické znalosti už nestačia a len on vedel, že bez následnej teórie by sme sa dopúšťali na pohľad nepatrných omylov, ale s obrovskými následkami. A verte či neverte, história mu vždy dala za pravdu. Robil to často, skoro s určitou pravidelnosťou, akoby medzi dvomi úrovňovými stupňami nášho odborného života.
A tento stav nastal teraz aj v našom seriáli. Z vašich mailov som zistil, že ste skutočne snaživí študenti a aj vaše projektíky naznačujú, že sa Slovensko báť nemusí o dostatok šikovných programátorov. Mnohí z vás sa púšťajú do rozsiahlejších (inak veľmi solídnych) projektov, a aby ste sa nedopustili tých nepatrných chýb s obrovskými následkami, nastal čas, aby sme si povedali niečo o koncepcii návrhu projektov. Už vás počujem - „Zase len teória!“ (to isté sme vravievali aj my!), ale až sa ňou prehryzieme, zistíme, že dokážeme robiť efektívnejšie projekty, odolnejšie voči systémovým chybám, stabilnejšie a flexibilnejšie voči požiadavkám v budúcnosti. Naozaj, nestraším vás, a skúste mi uveriť, tak ako sme aj my časom uverili slovám nášho učiteľa. A na okraj, tieto teórie budú obecne platné, teda neviažu sa ku konkrétnemu databázovému systému.

Relačný model
Vráťme sa naspäť k našim začiatkom. Hovorili sme si, že vzťahy medzi dátami sú postavené na relačnej algebre. Je to súbor základných matematických princípov odvodených z teórie množín a predikátovej logiky. Relačný model definuje spôsob, akým je možné dáta reprezentovať (štruktúru dát), spôsoby ich ochrany (integritu dát) a nakoniec operácie, ktoré môžeme nad dátami vykonávať (manipuláciu s dátami). Len pre úplnosť - toto dátové modelovanie zaviedol Dr.E.F.Codd a svoje výsledky zverejnil v roku 1970 (teda nič nové), ktorý sa stal neskôr výskumným pracovníkom firmy IBM.
Relačný model nepredstavuje jedinnú metódu spravovania dát. Existujú aj iné možnosti, ako hierarchický, sieťový alebo hviezdicový model. Každý má samozrejme svojich zastáncov aj odporcov a každý má pre určitú skupinu úloh rôzne prednosti. Relačný model je vďaka svojej vysokej efektivite a flexibilite veľmi obľúbený v realizácii databáz z bežného života.
Obecne sa dá povedať, že relačné databázové systémy vykazujú následné charakteristické vlastnosti:
- všetky dáta sa pomyselne dajú reprezentovať v pravidelne usporiadaných štruktúrach s riadkami a stĺpcami, ktorým hovoríme ***relácie***
- všetky hodnoty v databáze sú ***skalárne***. To znamená, že v každej konkrétnej pozícii riadku a stĺpca danej relácie sa nachádza jedna (presnejšie práve jedna) hodnota
- operácie v databáze sa vykonávajú vždy nad celou reláciou a ich výsledkom je opäť iná celá relácia; tomuto hovoríme ***uzáver***

Čo je to tá ***relácia***? Ak v nej poznávate niečo, čo je to čosi ako tabuľka, nie ste ďaleko od pravdy. Dr. Codd pri formulácii relačného modelu zvolil pojem ***relácia***, pretože ten bol voči iným pojmom relatívne „voľný“ a nemal iné, zavádzajúce významy ako napr. slovo ***tabuľka***. Reláciou môže byť čokoľvek, čo je usporiadané do štruktúry (formátu) riadkov a stĺpcov a čo obsahuje skalárne hodnoty. Existencia určitej relácie je teda úplne nezávislá na jej fyzickej reprezentácii (nezaujíma nás, ako a kde je fyzicky uložená na disku).

Základné pojmy
Najabstraktnejšou časťou celého návrhu databáze je ***datový model*** - to je totiž myšlienkový (pojmový) popis daného priestoru problému. Datové modely sa vyjadrujú pomocou ***entít***, ***atribútov***, ***domén*** a ***vzťahov***. Aby sme mohli pokračovať pri objasňovaní teórie a koncepcie databáz, vysvetlíme si základné pojmy, s ktorými budeme naďalej narábať:

Doména
Doména je množina hodnôt rovnakého významu. Doménou môže byť napríklad vek alebo priezvisko. Hodnoty v doméne sú rovnakého dátového typu - číslo, reťazec znakov, dátum a podobne. Je to ***obor hodnôt***, ktoré tvorí množina všetkých prípustných platných hodnôt, ktoré smie určitá položka obsahovať.
Aký je rozdiel medzi oborom hodnôt a dátovým typom?
Dátový typ je fyzický pojem, obor hodnôt je pojem logický: „číslo“ je dátový typ, zatiaľ čo „vek“ predstavuje už obor hodnôt, obsahujúci „čísla“.

A ešte jeden príklad: Položka „Priezvisko“ a položka „Adresa“ majú rovnaký dátový typ - je to reťazec znakov. Ale už z názvov je zrejmé, že budú mať iný obor hodnôt, takže náležia do rôznych domén.

Kartézsky súčin množín A, B
Majme množinu usporiadaných dvojíc *[x, y]* , pre ktoré platí, že *x* patrí do množiny **A** a *y* patrí do množiny **B**. Potom platí, že počet prvkov v kartézskom súčine je daný počtom prvkov v množine **A** *krát* počet prvkov v množine **B**.
Ak máme množinu *A = {1, 2, 3}* a množinu *B = {a, b},* potom je kartézsky súčin **M = A x B** daný takto:
M = {[1,a], [1,b], [2,a], [2,b], [3,a], [3,b]}
Príklad:
V našej knižnici máme dve množiny - množinu kníh, ktoré požičiavame a množinu čitateľov, ktorí si knihy požičiavaju. Kartézskym súčinom je spojenie týchto množín tak, že každá kniha by bola priradená jednému čitateľovi a zároveň by jeden čitateľ mal priradené všetky knihy. Ak máme 10 kníh a 5 čitateľov, kartézsky súčin by obsahoval 10 x 5 =50 záznamov. Spomeňme si, že sme takéto spojenie tabuliek už robili, a to v 8.časti seriálu.

Relácia
Už vieme, čo je to relácia. Matematické vyjadrenie je, že relácia je ľubovoľná podmnožina kartézskeho súčinu, napr: *R = {[1,a], [2,b], [3,a]},* čo je naozaj podmnožinou (čiastočkou) kartézskeho súčinu *M*. To, aké prvky bude relácia obsahovať, je definované práve konkrétnym SQL príkazom s určenými podmienkami.
Príklad:
Ak by SQL príkaz definoval vybratie tých čitateľov, čo splňujú určitú podmienku, napr. nemajú požičanú ani jednu knihu, vytvorila by sa podmnožina kartézskeho súčinu, ktorá vyhovuje tejto podmienke. Tak by vznikla relácia. Relácia môže byť trvalá (ako napr. tabuľka), odvodená (ako určitý pohľad na reláciu trvalú) alebo dočasná (len v pamäti počítača, napr. pri spojovaní tabuliek).

Entita
Entita je čokoľvek, o čom v systéme potrebujeme uchovávať potrebné informácie.
Pri zahájení prác na návrhu dátoveho modelu nie je zostavovanie prvotného zoznamu entít vôbec obtiažne. Ak si rozoberieme (sami, alebo s našim zákazníkom, pre ktorého projekt robíme) priestor problémov, používame podstatné mená ako vhodných kandidátov na entity. *„Autori píšu knihy“*. *„Vydavatelia vydávajú knihy“*. *„Dodávatelia dodávajú knihy“*. *„Čitatelia si požičiavajú knihy“*. Slová ako *autori*, *vydavatelia*, *dodávatelia*, *čitatelia* a *knihy* sú celkom isto entity.
Udalosti, ktoré v našom stručnom „rozhovore“ prezentovali slovesá *písať*, *vydávať*, *dodávať* a *požičiavať* sú **vzťahy** medzi entitami. Nie všetky však budeme v našom dátovom modeli potrebovať. To, že autori píšu, ba ani že vydavatelia vydávajú knihy nás netrápi, ale isto budeme chcieť evidovať vzťah medzi knihou a dodávateľom (*dodávať*), medzi knihou a čitateľom (*požičiavať*).
Entity môžu byť konkrétne - sú odrazom objektu vo fyzickom svete, napr. **kniha**, **čitateľ**, **autor** a pod. ale môžu byť aj abstraktné - modelujú myšlienky. Popisujú určitú vzájomnosť medzi rôznymi entitami. Príkladom môže byť napr. zodpovednosť konkrétneho pracovníka za istú oblasť činnosti firmy.
Veľmi zjednodušene môžeme povedať, že relácia je akási tabuľka o entite.

Atribúty
Vieme, že navrhovaný systém bude o každej entite zaznamenávať, sledovať a vyhodnocovať určité skutočnosti. Týmto skutočnostiam (údajom) sa hovorí ***atribúty danej entity***. Ak náš knižný systém obsahuje entitu **kniha**, chceme o nej viesť údaje o **názve**, **autorovi**, **vydavateľstve**, **dodávateľovi**, **žánri** a **cene**. Toto všetko sú atribúty. Nie vždy bývajú atribúty jednoznačné. Určovanie atribútov je sémantický proces. To znamená, že sa budeme rozhodovať podľa významu dát a podľa spôsobu ich využitia.
Zjednodušene povedané, atribúty sú v podstate stĺpce relácie (tabuľky).
Pozrime sa na jeden príklad - je ním adresa:
Stačí adresu modelovať ako entitu s jedným atribútom (*Adresa*), alebo je to vhodnejšie riešiť ako entitu s viacerými atribútmi (*Ulica, ČísloDomu, Mesto, Okres, PSČ*)? To záleží od požadovaného výsledku. Ak bude adresa slúžiť iba na korešpodenciu, stačí ju uvádzať ako jeden jedinný atribút, s ktorým sa iná činnosť okrem tlače nepočíta. Ak však budeme chcieť vyhodnocovať určitú štatistiku podľa okresov alebo dokonca ulíc, bude vhodnejšie evidovať adresu ako súbor atribútov *Ulica, ČísloDomu, Mesto, Okres, PSČ*.
Tu môžeme použiť dve stratégie:
Prvá stratégia znie takto: začnime požadovaným výsledkom a snažme sa neurobiť návrh zložitejší, než aký nutne musí byť. Dobrým spôsobom je klásť si otázky a na základe odpovede stanoviť požadovanú štruktúru. Stačí si položiť otázku: „Kam mám poslať čitateľovi poštu?“ a ak odpoveď vyhovuje, vyhovuje model s jedinným

atribútom. Ak však položíme otázku „V akom okrese daná osoba býva?“ , podľa odpovede vieme, že musíme nadefinovať inú štruktúru.
Nesmieme zabudnúť, že výsledný model musí byť natoľko flexibilný, aby dokázal zodpovedať nielen otázky, ktoré mu užívatelia budú klásť hneď teraz, ale tiež otázky, ktoré sa dajú do budúcna istým spôsobom predvídať. (A preto väčšina programátorov definuje adresu ako viacatribútovú entitu.). Ale pozor! Cenou za vysokú flexibilitu býva často zložitosť výsledného systému.

Druhá stratégia znie: hľadajme výnimky (výnimočné situácie). Musíme vedieť identifikovať všetky výnimky a systém musíme navrhnúť tak, aby dokázal zvládnuť čo najviac výnimiek bez zbytočného obťažovania užívateľa. Čo to znamená, si ukážeme na príklade mena:
V našich krajoch býva zaužívané, že každý občan má spravidla jedno meno (krstné) a jedno priezvisko. Ale existujú výnimky. Možno existuje pán, čo sa volá Augustus Maroš Kačka. A čo je teraz krstné meno? Augustus? Maroš? A priezvisko? Je to Kačka alebo Maroš Kačka?
Alebo taký Pavol Országh - Hviezdoslav? A čo taký Abúl Quásim Mansúr Firdausí?
Koľko môže byť takých výnimiek? A práve na odpovedi záleží, či evidenčný formulár na napĺňanie databáze bude obsahovať položky ako PrvéMeno, DruhéMeno, PrvéPriezvisko, DruhéPriezvisko, ŠľachtickýTitul alebo len klasický našský - Meno, Priezvisko?
Správne sa rozhodnúť je skutočne veľkou zodpovednosťou. Často treba preštudovať určité národné zvyky, napr. v ruských oblastiach majú osoby zásadne tri mená - rodné, otcovské a priezvisko, napr. Vladimír Iljič Lenin, ale veľmi často sa oslovujú len prvými dvoma - Vladimír Iljič.

Vzťahy
Okrem atribútov jednotlivých entít musíme v dátovom modeli určiť vzťahy, definované medzi rôznymi entitami. Z tvrdenia *„Čitatelia si požičiavajú knihy“* tak vyplýva existencia určitého vzťahu medzi entitami **čitatel** a **kniha**. Entity, zapojené do určitého vzťahu, sa nazývajú **účastníkmi**. Počet účastníkov označujeme ako **stupeň vzťahu**.
Drtivá väčšina vzťahov je *binárnych*, teda s dvomi účastníkmi - ako náš vzťah *„Čitatelia si požičiavajú knihy“*, ale nutné to zďaleka nie je. Bežné sú aj *ternárne* vzťahy, teda vzťahy medzi tromi účastníkmi. Z dvoch binárnych vzťahov *„Autori píšu knihy“* a *„Čitatelia si požičiavajú knihy“* vyplýva ternárny vzťah, vyjadrený vetou *„Autori píšu knihy pre čitateľov“*. Na základe pôvodných dvoch binárnych vzťahov nie sme schopní zistiť, ktorý autor napísal ktorú knihu pre ktorého čitateľa. To dokáže ošetriť až ternárny vzťah.

Kardinalita vzťahu
Pod **kardinalitou vzťahu** rozumieme počet výskytu objektov oboch entít, ktoré sa vzťahu účastnia.
Vzťah medzi ľubovoľnými dvomi entitami môže byť typu **1:1**, **1:n** alebo **m:n**.

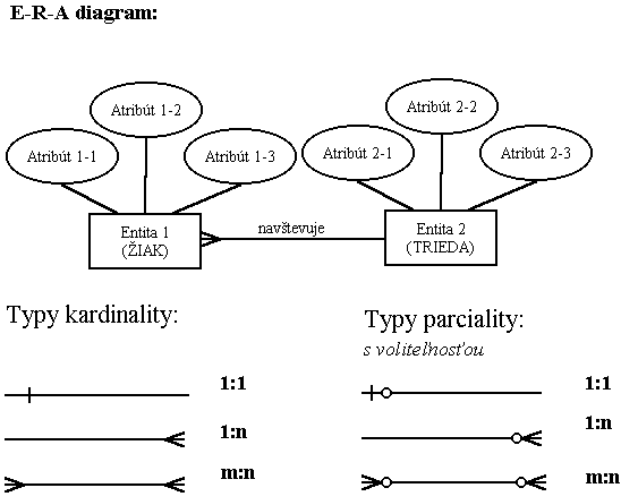
Vzťah **1:1** je vzťah, v ktorom na obidvoch stranách vystupuje iba jeden objekt danej entity. Tieto vzťahy sú v realite veľmi zriedkavé. Príkladom môže byť vzťah **manželia** medzi entitami **Muž** a **Žena**. V prípade monogamnej spoločnosti je zrejmé, že jedna žena má iba jedného muža a naopak, jeden muž má iba jednu ženu. Obdobným príkladom môže byť vzťah **triednictvo** medzi entitami **Učiteľ** a **Trieda**, kde učiteľ je triednym učiteľom iba v jednej triede a naopak, jedna trieda má iba jedného triedneho učiteľa.

Vzťah **1: n** (hovoríme mu aj *jedna k viacej*) býva v dátovom modeli najčastejší. Na jednej strane je jediný objekt entity, ktorý je vo vzťahu s jedným alebo viacerými objektami druhej entity. Ako príklad poslúži vzťah **čitateľ - kniha**, kde jeden čitateľ môže mať požičanú jednu alebo viac kníh, ale naopak, viacej kníh môže byť požičaných práve iba jedným čitateľom. Obdobne je to u vzťahu **trieda - žiak** (trieda ako inštitúcia, nie učebňa!), kde do jednej triedy (napr. 3.C) môže chodiť niekoľko žiakov, ale naopak, každý žiak z 3.C chodí iba do tejto jednej (jedinnej) triedy.

Vzťah **m:n** (nazývaný aj *viac ku viac*) je špecifickým vzťahom, v ktorom vystupuje viac objektov na obidvoch stranách. Napríklad: každý učiteľ vyučuje mnoho žiakov a každý žiak chodí na hodiny k mnohým učiteľom. Alebo vo vzťahu zamestnanec - úloha môže viacej zamestancov riešiť jednu úlohu a zároveň môže jeden zamestnanec riešiť viac úloh.

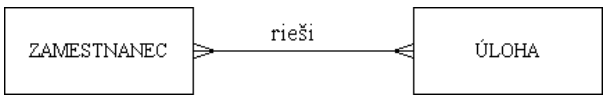
Parcialita vzťahu
Okrem *kardinality* vzťahu môžeme ešte rozlišovať *povinnosť* a *voliteľnosť* jeho existencie. Pozrime sa na to takto: Musí mať každá žena manžela a každý muž manželku? Musí byť každý učiteľ triednym učiteľom? Vidíme, že sa môžu vyskytovať typy vzťahov, ktoré nemusia existovať u všetkých objektov danej entity - existujú predsa slobodné ženy a slobodní muži a učitelia bez triedníctva.

E-R a E-R-A diagramy
Model entít a vzťahov, ktorý popisuje dáta ako entity, atribúty a vzťahy medzi nimi, zaviedol poprvýkrát Peter Pin Shan Chen v roku 1976. Súčasne navrhol metódu jeho zobrazenia do diagramov, ktoré pomenoval ***diagramy entít a vzťahov*** (Entity - Relationship Diagram), ktoré poznáme pod skratkou ***E-R diagramy***. Ak v E-R diagramoch uvádzame aj atribúty entít, hovoríme o ***E-R-A diagramoch*** (Entity - Relationship - Attribute Diagram). V E-R diagramoch sa entity označujú pomocou obdĺžnikov, atribúty pomocou elipsy alebo oválov a vzťahy sa znázorňujú spojovaciou čiarou medzi entitami. Keďže ani najlepšie nakreslené diagramy nedokážu popísať všetky situácie z reálneho sveta, je nutné doplniť každý diagram slovným popisom. Na obrázku č. 14-1 je zoznam najzaužívanejších značiek v E-R-A diagramoch:

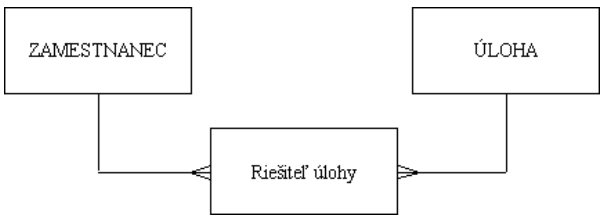


Všimnime si „vidličky“ na strane **žiaka** vo vzťahu k **triede**. Vyjadruje kardinalitu vzťahu žiaka a triedy. Takisto vidíme, že je možné znázorniť particialitu vzťahu - prázdny krúžok vyjadruje voliteľnosť na strane entity, ktorá nemusí existovať. Hovorím, že je to možné, ale nie povinné. Vzťah **1:n** akosi automaticky predpokladá výskyt *0* (nula) až *n* objektov.

Dekompozícia vzťahu m:n
Vzťah **m:n** je z hľadiska ďalšej práce veľmi komplexný a je nutné pokúsiť sa o jeho zjednodušenie. Nerobíme tak kvôli jeho implementácii na ďalšej, logickej úrovni návrhu., ale i pre prípad, že v tomto vzťahu je „ukrytá“ ďalšia entita, ktorá zatiaľ našej pozornosti unikla. Súčasťou tejto entity môžu byť aj atribúty, o ktorých sme tušili, že existujú, ale sme nevedeli, ku ktorej entite ich priradiť.
Dekompozíciou vzťahu m:n rozumieme vytvorenie novej, tzv. *väzbovej* entity, ktorá bude mať vzťahy **1:n** na obidve pôvodné entity vzťahu **m:n**. Pozrime sa na obrázok č. 14-2, čo je E-R diagram vzťahu *m:n* medzi entitami **Zamestanec** a **Úloha**:



Na obr. č. 14-3 je dekompozícia tohto vzťahu:



Novo vzniknutá entita **Riešiteľ úlohy** vyjadruje fakt, že zamestnanec môže byť naraz riešiteľom viacerých úloh a jedna úloha môže byť riešená naraz viacerými riešiteľmi.
Súčasťou tejto entity možu byť atribúty, ktoré nemožno priradiť k žiadnej z entít **Zamestnanec** a **Úloha**. Príkladom je atribút popisujúci hodnotenie zamestnanca za odvedenú prácu na danej úlohe. Pretože zamestnanec môže pracovať na viacerých úlohách a za každú môže byť hodnotený inak, nemôže byť tento atribút umiestnený do entity **Zamestnanec**. Zároveň nemôže byť umiestnený do entity **Úloha**, pretože na jednej úlohe môžu pracovať viacerí zamestnanci, každý s iným úspechom. Jediným správnym umiestnením atribútu **Hodnotenie** je do entity **Riešiteľ úlohy**, pretože sa vzťahuje vždy k dvojici *{zamestnanec, úloha}.*

Tri úrovne návrhu
Návrh projektu a jeho štruktúry by nemal byť, zvlášť u rozsiahlejších systémov, živelným procesom postupne reagujúcim na vzniknuté požiadavky. V súčasnosti existujú historicky overené postupy a pravidlá návrhu, ktoré umožnia a do istej miery aj zaručia vytvorenie kvalitnej dátovej základne, ktorá bude riadne zadokumentovaná, logicky konzistentná a bude umožňovať relatívne jednoduché zapracovanie skutočností, vzniknutých neskôr. Najdôležitejšia je počiatočná analýza oblasti, ktorú chceme v projekte zobraziť, ešte predtým, ako začneme vytvárať tabuľky a SQL príkazy. Čím neskôr totiž odhalíme chyby v návrhu dátovej základne, tým väčšiu námahu a tým pádom aj finančné prostriedky budeme musieť obetovať na ich nápravu.
Jedným z možných postupov je oddelenie popisu oblasti našeho záujmu od vlastnej implementácie na počítači. Spôsob implementácie môže výrazne ovplyvniť štruktúru dátovej základne, ale s oblasťou, ktorú dátová základňa popisuje, nemá nič spoločné. Princíp troch úrovní rozdeľuje postup návrhu datovej základne na tri kroky, ktoré si popíšeme.

Konceptuálna úroveň
Na tejto úrovni sa snažíme popísať predmetnú oblasť pomocou všetkých entít, ktoré sa v nej vyskytujú a všetkých vzťahov medzi týmito entitami. V žiadnom prípade v tejto fáze neberieme do úvahy neskorší spôsob implementácie a do istej miery ani neskoršie obmedzenia technologického charakteru. Týmto môžeme venovať všetku energiu na pochopenie vlastného problému. Nakoniec získame aj obecne platný popis danej oblasti, ktorý môžeme použiť pre implementáciu v odlišných databázových systémoch bez nutnosti opätovnej analýzy.
Tu sú ciele konceptuálneho modelu:
- vytvoriť obraz reality vo formalizovanej podobe, nezávislý na neskoršom spôsobe implementácie
- formalizovať požiadavky užívateľov a dať návrhárom ľahko pochopiteľný prostriedok pre komunikáciu s užívateľom, ktorému budú aj užívatelia rozumieť
- vytvoriť podklad pre návrh dátovej základne

Výsledkom tejto úrovne by mali byť E-R diagramy so slovným popisom zobrazovanej situácie. Predchádzajú im rozsiahle debaty so zadávateľom projektu.
Spokojne sa môže stať, že zadávateľom, projektantom, programátorom, užívateľom a správcom projektu je tá istá osoba - teda my. To však neznamená, že by sme nemohli všetky príslušné debaty a hádky vykonať sami so sebou. (a nie je to duševná porucha). Stačí sa vždy postaviť do jednej z uvedených rolí a nazerať na projekt z iného uhlu. Najkritickejšia je rola užívateľa - ten máva najviac nepríjemných otázok, a preto nebuďme ako programátori ješitní, aby sme sa v tejto "jednoroli" nezhnusili sami sebe!
Osvedčilo sa mi, že pri tvorbe projektu som chvíľu pracoval s budúcim užívateľom. Zistil som jeho zvyky, návyky a názory. Odmenou mi bolo, že sa tento užívateľ neskôr na projekt nesťažoval.

Ak nemáme o oblasti budúceho projektu ani "páru", požiadajme zodpovedného pracovníka, aby nám ukázal svoju činnosť "v tužke", teda ako pracuje doteraz s papierovou formou agendy. Čo píše, čo maže, ako to spočitáva, ktoré informácie sú pre neho vstupy a ktoré on produkuje ako svoje výstupy.

Prax hovorí, že tejto oblasti treba venovať asi 70 % celkových nákladov na projekt. (Myslím tým nielen peniaze, ale aj úsilie, čas a iné nefinančné prostriedky).

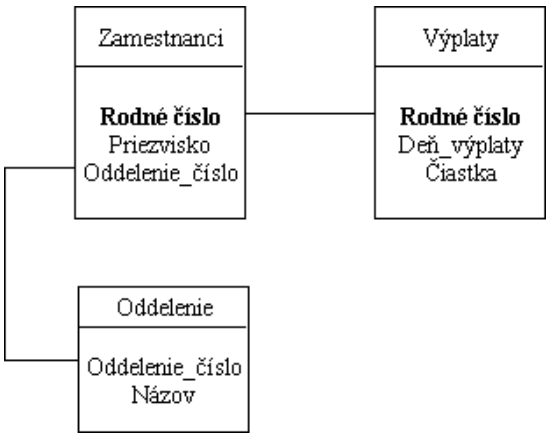
Ak toto všetko máme úspešne za sebou, pristúpime k logickej úrovni návrhu.

Logická úroveň
Pre popis dát na logickej úrovni sa v relačných databázach používa tzv. **relačná schéma**. Relačná schéma obsahuje tabuľky vrátane všetkých ich stĺpcov. V schéme sú vyznačené ***primárne kľúče*** v tabuľkách, ale aj ***cudzie kľúče*** ako odkaz na primárne kľúče v inej tabuľke. Tento odkaz je väčšinou vyznačený ako čiara spojujúca stĺpce v dvoch tabuľkách. Súčasťou relačnej schémy možu byť aj popisy integritných omedzení tabuliek a stĺpcov.

Prevod konceptuálneho modelu dát na relačnú schému je možné skoro automatizovať pomocou nasledujúcich pravidiel:

1) Každá entita v konceptuálnom modeli sa stáva samostatnou tabuľkou.
2) Identifikátor entity sa stáva primárnym kľúčom
3) Ak je súčasťou vzťahu jeden alebo viac atribútov, je nutné vytvoriť novú (väzbovú) entitu podobne ako u vzťahu *m:n* a z nej vytvoriť tabuľku.
4) U každého vzťahu zvolíme tabuľku, ktorá bude obsahovať cudzí kľuč ako odkaz do inej tabuľky. Pre voľbu tabuľky použijeme tieto pravidlá:
   a) Ak je vzťah typu *1:n*, bude cudzí kľúč pridaný do tabuľky na strane *n*.
   b) Ak sa jedná o parciálny vzťah *1:1*, bude cudzí kľúč pridaný do tabuľky, ktorá sa vyskytuje vo vzťahu nepovinne.
   c) Ak sa jedná o neparciálny vzťah *1:1*, volíme tabuľku, ktorá je vecne "podriadená" (napr. vo vzťahu *zamestnanec - učiteľ* to bude učiteľ, lebo je špecializovanou formou (podmnožinou) zamestnanca).
   d) Vzťahy typu *m:n* najprv dekomponujeme na dva vzťahy *1:n* a potom postupujeme podľa pravidiel pre tento typ vzťahov.
5) Ak sa vyskytne v konceptuálnom modeli špecializácia, máme niektorú z týchto možností:
   a) Vytvoríme jednu tabuľku (napr. Zamestnanec), ktorá bude obsahovať stĺpce pre všetky atribúty, vrátane tých, čo sa vyskytujú len u špecializovaných entít (napr. učiteľov). Na každom riadku zostanú niektoré stĺpce nevyplnené (budú obsahovať hodnotu *NULL*). Tento spôsob je najmenej náročný z hľadiska tvorby SQL dopytov, ale je nehospodárny miestom, ktoré budú dáta zaberať, a spracovávanie dopytov nad takto vytvorenou tabuľkou bude trvať dlhšie.
   b) Vytvoríme zvláštnu tabuľku pre každú špecializovanú entitu. Budeme mať teda tabuľky **Učitelia**, **Sekretárky**, **Študijní_Referenti** atď. Nebudeme sice plýtvať miestom, ale bude nám činiť potiaže práca so všetkými zamestnancami naraz - púhe vypísanie menného zoznamu, zvýšenia platu a podobne. Veľmi ľahko sa môže stať, že budeme nútení pridať ďalšiu tabuľku (napr. **Externisti**) a zabudneme opraviť niektorý z už vytvorených dopytov, pracujúci so všetkými zamestnancami.
   c) Vytvoríme tabuľku **Zamestnanci**, ktorá bude obsahovať stĺpce spoločné pre všetky typy zamestnancov. Pre každú špecializáciu (entitu) potom vytvoríme tabuľku ďalšiu, ktorá bude obsahovať stĺpce špecifické pre túto entitu. Tento spôsob spojuje výhody obidvoch predchodzích. Musíme však zaistiť referenčnú integritu dát medzi špecializovanými tabuľkami a tabuľkou hlavnou.

Na obr. č. 14-4 je ukážka jednoduchej relačnej schémy, popisujúcej tri tabuľky - **Zamestnanci**, **Výplaty** a **Oddelenia** a ich vzájomné vzťahy. Tá by mala byť výsledkom tejto úrovne návrhu:



Na záver definujeme aj príkazy na získavanie a manipuláciu s dátami v jazyku SQL a pristúpime k implementačnej úrovni.

Implementačná úroveň

Na implementačnej úrovni vyberáme konkrétny databázový systém, v ktorom vytvoríme dátovu základňu. Po jeho výbere môžeme začať využívať aj rôzne neštandardné funkcie zvoleného prostredia. Ich použitie by sme však mali dôsledne zvážiť, zvlášť kvôli možnému neskoršiemu prechodu na iný databázový systém. Z hľadiska jazyka SQL je nutné na tejto úrovni vziať do úvahy aj možné dielčie odlišnosti v príkazoch, zvlášť v skupine príkazov pre definíciu dát.

Nabudúce sa budeme venovať normalizácii relácií. Že ste to ešte nepočuli? Tak to je na databázach asi tá najdôležitejšia časť teórie.

Malé veľké databázy II - 5.časť

Dobrá správa! Práve vyšla MySQL verzia 4.0!!! Teda, zatiaľ iba v alfa release, preto treba s ostrým nasadením ešte počkať. A čo je v nej nové?

- Optimalizácia MYSQL kódu prináša zvýšenie rýchlosti hlavne v oblasti: viacnásobné INSERTy, vyhľadávanie, vytváranie FULLTEXT- ových indexov ako aj COUNT(DISTINCT).

- Transakcie a zamykanie tabuliek na úrovni riadkov sú implicitnou súčasťou databázového stroja (systém InnoDB).

- MySQL podporuje zabezpečenú komunikáciu medzi klientom a serverom na úrovni SSL protokolu, čo znamená možnosť umiestniť DBS server aj mimo zabezpečenú zónu (napríklad geograficky pred firewall).

- Úpravy pre triedenie nemeckých jazykových sád

- Podpora autoinkrementácie alá SYBASE (IDENTITY) a import databáz vrátane TRUNCATE TABLE (ako v Oracle).

- Podpora UNION-ov (spojenie viacerých SELECTOV)

Zároveň autori oznamujú, že sa pripravuje verzia 4.1, ktorá bude obsahovať aj subselekty, uložené procedúry a ďalšie vylepšenia. Túto verziu môžeme očakávať až začiatkom budúceho roka.

Vráťme sa k našej milovanej teórii.

Zopakovanie

Vieme, čo je entita, doména a relácia. Vieme, že každá relácia obsahuje entitu, ktorá má konkrétne atribúty. Poznáme aj vzťahy medzi entitami, ich kardinalitu a parcialitu, vieme ich prípadne dekomponovať. Dokážeme nakresliť E - R diagram, z ktorého vytvoríme relačnú schému. Nakoniec definujeme tabuľky a SQL príkazy na prácu s nimi.

Dnes si povieme niečo o funkčnej závislosti a vysvetlíme si normalizáciu relácií.

Funkčná závislosť

Aby sme sa vyhli zložitej matematike, iba povieme, že funkčná závislosť je tvrdenie o reálnom svete. Napríklad plat zamestnanca závisí na tom, akú funkciu zamestnanec vykonáva, teda plat zavisí od funkcie, zapisujeme FUNKCIA - PLAT. Druhým príkladom je cena cestovného, ktorá závisí od dĺžky precestovanej trasy, teda DĹŽKA_TRASY - CENA_CESTOVNÉHO. Podobných príkladov by sme našli v reálnom živote niekoľko.

Študijný príklad

V minulej časti sme si povedali niečo o troch úrovňach návrhu - konceptuálnej, logickej a implementačnej. Aby sme si to trochu objasnili a doplnili o nové znalosti prakticky, vytvoríme si tento študijný príklad:

V nemenovanej štátnej inštitúcii je archív hudobných cédéčiek. Slečna referentka (ktorá je naša kamarátka, lebo je veľmi fajn) sa v tom už nedokáže vyznať a tak sme jej slúbili, že pre ňu vytvoríme program, ktorý jej bude tieto cédéčka evidovať. Keďže sa iba archivujú, a sú iba po jednom kuse z každého, nie je potrebné riešiť výpožičky, náklady ani nič podobné.

Na prvý pohľad sa zdá táto úloha veľmi triviálna. Ale pozrime sa na to postupne.

Konceptuálna úroveň

No, entitu by sme mali. Pomenujeme ju **CD**. A čo budeme o nej evidovať? Hádam názov, meno kapely, názov vydavateľa a zoznam pesničiek. Takže vytvoríme reláciu s požadovanými atribútami, tak ako je to na obr. 15 - 1:

Táto relačná schéma je jednoduchá, však?Ale ešte sme neskončili. V skutočnosti, sme iba na začiatku. Máme síce pekný diagram pre náš projekt, ale cítime, že to ešte nie je to pravé orechové. Nastal čas zaoberať sa normalizáciou.

Normálne formy

Normalizácia je činnosť, ktorá vedie k dobre navrhnutým tabuľkám. Princípy normalizácie definoval nám už známy E.F. Codd a nie je to suchá teória. Boli overené praxou a za tie dlhé roky tvorby databázových aplikácii sa vypracovali určité postupy tvorby tabuliek, ktoré sa nazývajú normálne formy. Najčastejšie sa používajú prvé tri normálne formy, ale existujú aj ďaľšie, ako si spomenieme neskôr.

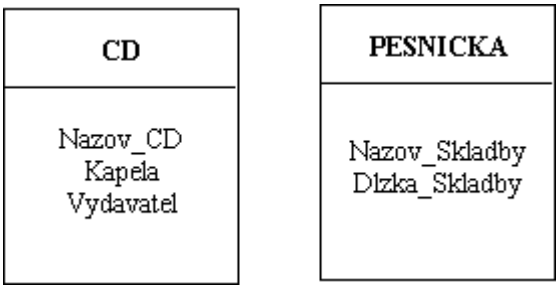
Vráťme sa k nášmu príkladu a pokračujme cestou normalizácie. (Pre tých skôr narodených - prosím neasociovať so 70. rokmi!)

Prvá normálna forma (1NF)

O relácii (tabuľka) hovoríme, že je v prvej normálnej forme, ak sú všetky jej atribúty atomické, t.j. ďalej nedeliteľné. Inými slovami - každý atribút relácie môže mať iba jednu hodnotu, teda nemôže byť jeho hodnotou ďaľšia relácia.

Pozrime sa ešte raz na našu tabuľku. Vidíme, že nespĺňa ani prvú normálnu formu, lebo atribút **Pesničky** nedosahuje iba jednu hodnotu, ale môže ich mať viac - podľa počtu skladieb na konkrétnom cédečku. Predstvame si, že by naša kamarátka zrazu potrebovala zistiť, na ktorom disku sa nachádza určitá skladba. Ak by sme ponechali tabuľku v tejto forme, mali by sme pri tvorbe selektu, ktorý by vyhľadal konkrétnu skladbu, asi značné problémy.

Tušíme, že atribút **Pesničky** obsahuje ďaľšie atribúty ako názov alebo dĺžka skladby. To znamená, že by to mohla byť samostatná entita s týmito dvoma atribútmi. Preto prevedieme rozloženie na dve samostatné entity, ktoré budú tvoriť dve samostatné tabuľky (obr.č.15 - 2):



Teraz sú ***Nazov_Skladby*** a ***Dlzka_Skladby*** atribúty entity **PENICKA**, takže datový model je v prvej normálnej forme (1NF). Nanešťastie, ešte nemáme popísané vzťahy medzi oboma entitami. Na to potrebujeme zadefinovať akýsi ***unikátny identifikátor.***

Unikátny identifikátor

Každá entita potrebuje mať svoj identifikátor. Môžeme ho nazvať aj **ID**. Bude to nový atribút entity, ktorý bude mať tieto vlastnosti:

- bude unikátny v celej tabuľke, popisujúcej danú entitu. To znamená, že sa v celej tabuľke nenájdu dva riadky, ktoré by mali rovnakú hodnotu tohto atribútu.

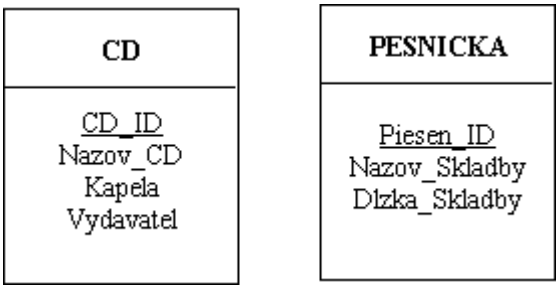
- jeho hodnota nesmie byť prázdna (NULL) po celú dobu existencie tabuľky

- už raz zadaná hodnota tohto identifikátoru sa nesmie v danom riadku tabuľky nikdy zmeniť po dobu existencie tabuľky.

My už v podstate tento identifikátor poznáme. Je ním nám dobre známy ***primárny kľúč***. Vraťme sa k začiatkom seriálu a zopakujme si jeho podstatu. Základom úspechu je správne zvoliť primárny kľúč. Začiatočníci robia často chybu v tom, že za primárny kľúč vyberú napr. priezvisko. My však vieme, že existujú ľudia, ktorí majú rovnaké priezvisko. Existuje pravidlo, ktoré hovorí, že primárny kľúč by mal byť čo najmenší. Preto je veľmi vhodné zvoliť ako primárny kľúč číslo.

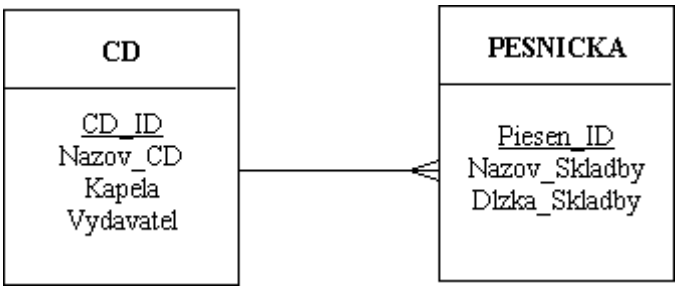
Primárny kľúč môže byť aj zložený z viacerých atribútov - to závisí od konkrétnej aplikácie.

Upravíme diagramy tak, že do každej entity vložíme identifikátor, ktorý bude primárnym kľúčom tabuľky (obr.15 - 3):



Primárne kľúče budeme označovať s príponou **ID** a v diagrame budú zvýraznené potrhnutím.

Aby bola schéma úplná, musíme nadefinovať **relačné vzťahy**. Ak to chceme vyjadriť slovne, môžeme povedať, že na jednom cédečku môže byť jedna alebo viac pesničiek. Z teórie o vzťahoch je zrejmé, že sa v našom prípade jedná o vzťah 1: n. Tento vzťah zakreslíme pomocou vidličky tak, ako je to na obr.č.15 - 4:

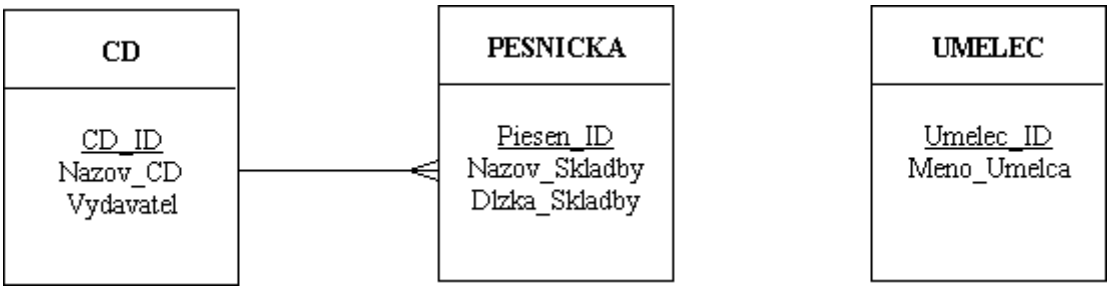


Prvá normálna forma je kompletná.

Druhá normálna forma (2NF)

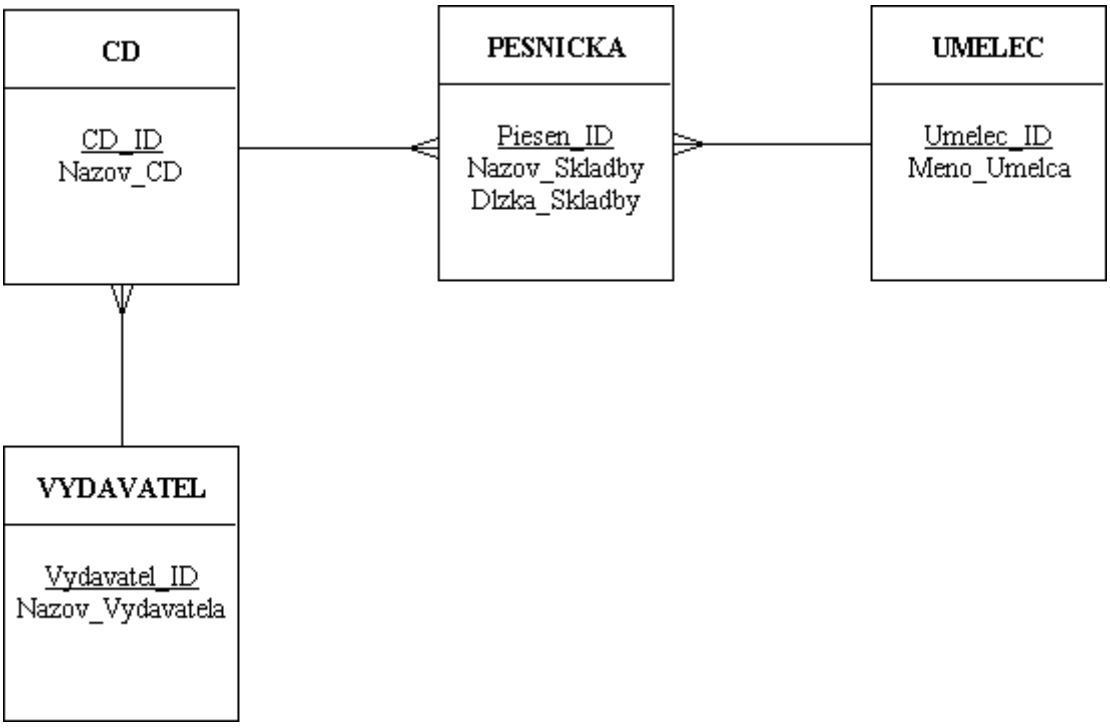
Tabuľka je v druhej normálnej forme, ak je v prvej normálnej forme a naviac každý atribút, ktorý nie je primárnym kľúčom je na primárnom kľúči úplne závislý.

Pozrime sa na náš datový model. V tabuľke **CD** je atribút ***Kapela***, ktorý určite nie je závislý na primárnom kľúči ***CD_ID***, lebo sa môže nachádzať na rôznych diskoch. Čo vlastne atribút ***Kapela*** popisuje? Popisuje hudobnú skupinu, alebo všeobecnejšie - umelca. A umelec je už samostatná entita so svojími atribútmi. Preto prevedieme znova dekompozíciu schémy, kde vyjmeme atribút ***Kapela*** a nahradíme ho novou entitou s atribútmi ***Umelec_ID*** ako primárnym kľúčom a ***Meno_Umelca*** ako ďalším atribútom (obr.č.15 - 5):



Aké budú vzťahy medzi entitami **CD**, **PESNICKA** a **UMELEC**? Pre jednoduchosť predpokladajme, že jeden umelec môže naspievať viac pesničiek, ale každá pesnička je naspievaná iba jedným umelcom. V takom prípade sa jedná o vzťah 1: n, ktorý zadefinujeme „vidličkou“ (obr.č.15 - 6):

Ešte stále však nie sme v druhej normálne forme. Prečo? Pretože atribút ***Vydavatel*** je podobný problém ako atribút ***Kapela***. Preto dekomponujeme atribút ***Vydavatel*** na samostatnú entitu s jej parametrami tak, ako je to na obrázku č.15 - 7:



Konečne máme náš datový model v druhej normálnej forme.

Tretia normálna forma

O relácii hovoríme, že je v tretej normálnej forme, ak je v druhej normálnej forme a zároveň všetky jej atribúty, ktoré netvoria primárny kľúč, sú na sebe nezávislé. Ak by existoval atribút, ktorý je závislý na inom atribúte, musíme ho presunúť do novej entity.

Ak sa pozrieme na náš datový model a predpokladáme, že nás už bližšie nezaujímajú ďalšie informácie o vydavateľstve, môžeme povedať, že všetky atribúty v jednotlivých entitách sú na sebe navzájom nezávislé. (Ak by nás zaujímala aj presná adresa vydavateľstva, museli by sme v zmysle prvej normálnej formy rozpracovať nové atribúty entity **VYDAVATEL**, ako ulica, mesto, PSČ a iné). Takže môžeme povedať, že tento datový model splňuje túto podmienku a je v tretej normálnej forme.

Logická úroveň návrhu

Teraz máme kompletný logický návrh dátového modelu. Zhrňme si zásady postupu:

- identifikácia a modelovanie entít

- identifikácia a modelovanie vzťahov medzi entitami

- identifikácia a modelovanie atribútov

- identifikácia a stanovenie identifikátorov (primárnych kľúčov) pre každú entitu

- normalizácia

Datový model, ktorý sme vytvorili, je skutočne jednoduchý. Zatiaľ je to všetko v teoretickej rovine. Aby bol náš projekt aj funkčný, musíme pokročiť do implementačnej úrovne.

Implementačná úroveň

Ako už vieme, v tejto úrovni pripravujeme datový model na konkrétne technické prostriedky. Po túto úroveň bola činnosť pre všetky druhy SQL serverov rovnaká. Implementácia je vlastne preklad dátového modelu do konkrétneho SQL jazyka. Teraz pristúpime - na základe doterajších skúseností - k implementácii dátového modelu na server MySQL.

Aby sme sa nedopustili chýb, popíšme si základné pravidlá implementácie:

1) Entity sa stávajú tabuľkami vo fyzickej databáze
2) Atribúty sa stávajú stĺpcami v tabuľke. Musíme zvoliť príslušný dátový typ pre každý stĺpec
3) Unikátne identifikátory sa stávajú primárnymi kľúčmi v tabuľke.
4) Vzťahy sú modelované ako cudzie kľúče. (Objasníme neskôr.)

Ak budemem aplikovať tieto pravidlá , dostaneme popis fyzickej databáze, ako je to v tab.č. 15 - 8:

Tab.č.15 - 8: Fyzická implementácia dátového modelu:

Tabuľka	Stĺpec	Dátový typ	Poznámka
CD	CD_ID	INT	primárny kľúč
	Nazov_CD	VARCHAR(50)	
PESNICKA	Piesen_ID	INT	primárny kľúč
	Nazov_Skladby	VARCHAR(50)	
	Dlzka_Skladby	TIME	
UMELEC	Umelec_ID	INT	primárny kľúč
	Meno_Umelca	VARCHAR(50)	

VYDAVATEL	Vydavatel_ID	INT	primárny kľúč
	Nazov_Vydavatela	VARCHAR(50)	

V tabuľke č.15 - 8 sú definované kroky 1 až 3 implementácie. Všimnime si, že sme primárne kľúče deklarovali ako *INT* - my už vieme prečo. Stĺpec ***Dlzka_Skladby*** bude uchovávať časové informácie o dĺžke každej skladby. Aj keď nepredpokladáme, že by sme niekedy pracovali s týmto časovým údajom, predsa sme ho deklarovali ako dátový typ *TIME*. Ale spokojne sme ho mohli deklarovať aj typu *VARCHAR*. Ostatné stĺpce sme deklarovali typu *VARCHAR* o maximálnej dĺžke 50 znakov, čo by mohlo vyhovovať.

Ešte sme neimplementovali vzťahy. Tie sa definujú ako pridanie cudzích kľúčov do tabuľky, ku ktorej vzťah smeruje. Cudzí kľúč - *foreign key* - je primárny kľúč tabuľky na druhej strane vzťahu.

Takže ak sa dobre pozrieme na obr.č. 15 - 7, z ktorého pri implementácii vychádzame, musíme vložiť:

- stĺpec ***Vydavatel_ID*** do tabuľky **CD**

- stĺpec **CD_ID** do tabuľky **PESNICKA**

- stĺpec ***Umelec_ID*** do tabuľky **PESNICKA**

Jednoducho povedané, kópiu primárneho kľúča každej tabuľky presunieme v smere „vidličky vzťahu“ do druhej tabuľky.

Takto vytvoríme úplnú implementáciu dátového modelu (tab.č.15 - 9):

Tab.č.15 - 9: Úplná implementácia dátového modelu:

Tabuľka	**Stĺpec**	**Dátový typ**	**Poznámka**
CD	CD_ID	INT	primárny kľúč
	Nazov_CD	VARCHAR(50)	
	Vydavatel_ID	INT	cudzí kľúč
PESNICKA	Piesen_ID	INT	primárny kľúč
	Nazov_Skladby	VARCHAR(50)	
	Dlzka_Skladby	TIME	
	CD_ID	INT	cudzí kľúč
	Umelec_ID	INT	cudzí kľúč
UMELEC	Umelec_ID	INT	primárny kľúč
	Meno_Umelca	VARCHAR(50)	
VYDAVATEL	Vydavatel_ID	INT	primárny kľúč
	Nazov_Vydavatela	VARCHAR(50)	

Teraz máme kompletnú schému fyzickej databáze. Aby to bolo úplné, pomenujme túto databázu **ARCHIV**.

Nakoniec zostáva už len napísať skript v SQL jazyku, ktorý zabezpečí vytvorenie databáze **ARCHIV** s príslušnými tabuľkami na MySQL serveri (Výpis č.15 - 10):

```
Create Table CD (CD_ID INT NOT NULL,
•
                NAZOV_CD VARCHAR(50),
                VYDAVATEL_ID INT,
                Primary Key (CD_ID));
Create Table PESNICKA PIESEN_ID INT NOT NULL,
                      NAZOV_SKLADBY VARCHAR(50),
                      DLZKA_SKLADBYC
                      CD_ID INT,
                      UMELEC_ID INT,
                      Primary Key (PIESEN_ID));
Create Table UMELEC (UMELEC_ID INT NOT NULL,
                     MENO_UMELCA VARCHAR(50),
                     Primary Key (UMELEC_ID));
Create Table VYDAVATEL (VYDAVATEL_ID INT NOT NULL,
                        NAZOV_VYDAVATELA VARCHAR(50),
                        Primary Key (VYDAVATEL_ID));
```

Samozrejme nesmieme zabudnúť najpr vytvoriť databázu **ARCHIV**, napríklad cez program *mysqladmin*.

Ostatné normálne formy

Aby nás nezaskočila informácia, že existujú aj iné normálne formy, tak si ich iba spomenieme, že sú. Za treťou nornálnou formou nasleduje tzv. ***Boyce/Coddova normálna forma***.Je to určitá variácia tretej formy. Používa sa iba pri reláciách s viacerými kandidátnymi kľúčmi.

Ešte existuje ***štvrtá normálna forma***, ktorej základom je zákaz spojovania nezávislej opakovanej skupiny a konečne ***piata normálna forma***, ktorá sa týka tzv. spojenej závislosti.

Ale to už je vysoká škola teórie databáz a v praxi vystačíme s prvými tromi normálnymi formami.

Sme na konci s teoretickým výkladom, to aby som vás dlhšie nenudil. Prosím, vezmite si túto teóriu k srdcu a vyhnete sa zbytočným problémom. Tí, čo si myslia, že sú to len zbytočné blbosti (aj ja som bol taký!!!), mi dajú za pravdu neskôr.

Teraz už len zostáva navrhnúť to správne technologické okolie, aby sme nemuseli nútiť našu kamarátku pracovať v príkazovom riadku, a takto sa vraciame od teórie naspäť k praxi k navrhovaniu pekného oknoidného prostredia.

Nabudúce budeme pokračovať tam, kde sme pred týmto intermezzom skončili. Ukážeme si, ako dostaneme údaje z databáze na SQL serveri napr. do Excelu. Že to nejde? No veď uvidíme!

Malé veľké databázy II / 6.časť

Vráťme sa na chvíľu k 13.časti seriálu. Hovorili sme si o možných spôsoboch, ako sa dá pristupovať k dátam na SQL serveri. Rozobrali sme si jednotlivé vetvy A, B a C. Vetva „A“ rieši priame pripojenie pomocou front-end aplikácie, ktorou je napr. nám dobre známy mysql monitor. Touto vetvou sme sa zaoberali na začiatku. Vetva „B“ popisuje pripojenie pomocou PHP a HTTP servera, čo sme riešili nedávno. Vetva „C“ sa zaoberá prístupom k dátam, ktoré sú ďalej spracovávané konkrétnou natívnou aplikáciou. Tou môže byť napr. aplikácia napísaná v niektorom vyššom programovacom jazyku, ktorý umožňuje používať SQL príkazy, napr. Delphi, Visual Basic a pod.. Ale takou aplikáciou môže byť aj MS Excel. A tak si dnes ukážeme, ako sa dajú dáta, uložené niekde na inom, hoci aj vzdialenom MySQL serveri bežiacom na Linuxe, nádherne pripojiť do listu programu MS Excel. A aby to bolo efektnejšie, vytvoríme si z takto získaných dát aj pekný graf. Ja viem, že dnes asi pravých linuxákov nezaujmem, no ale povedzte, nepoteší sa takému výsledku váš šéf?

ODBC

Povedali sme si, že natívna aplikácia môže komunikovať s (ľubovoľným) SQL serverom dvojakým spôsobom:

- priamo, teda využívaním API funkcií daného SQL servera
- nepriamo, pomocou ODBC

Pre ilustráciu si predstavme aplikáciu MS Excel, ktorá vie (v spolupráci s MS Query, ktorá je súčasťou) zadávať SQL príkazy. Z pohľadu Excelu mu je úprimne jedno, na ktorý SQL server sa má pripojiť, pretože on používa bežné normalizované SQL príkazy. To, kam sa pripojí, mu zabezpečí určitý prostredník, a tým je **ODBC**. ODBC – *Open DataBase Conectivity* je určitý štandard, ktorý zabezpečuje ľubovoľnej aplikácii spojenie a používanie SQL príkazov. Každý výrobca SQL servera produkuje aj vlastný ODBC driver, ktorý zabezpečí komunikáciu aplikácie práve len s jeho serverom.

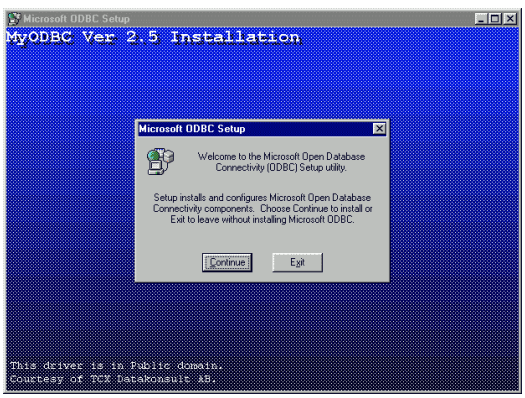
MyODBC

Preto aj MySQL server má ODBC driver (pre Win95/98 a WinNT). Jeho úlohou je prevziať od natívnej aplikácie SQL príkazy a pretransformovať ich do takej podoby, aby im rozumel MySQL server.
Ak sme pozorne čítali, zistíme, že ak budeme v našej aplikácii, vytvorenej napr. v Delphi, Visual Basicu alebo dokonca v MS Accessu používať štandardné SQL príkazy, môžeme jednoducho tú istú aplikáciu použiť v spojení s ľubovoľným SQL serverom. Stačí, ak použijeme príslušný ODBC driver (ovládač) daného SQL servera. Túto výhodu používajú mnohí programátori, lebo nemusia pracne upravovať svoje aplikácie.
My sa však vrátime k nášmu MySQL serveru.

Inštalacia MyODBC

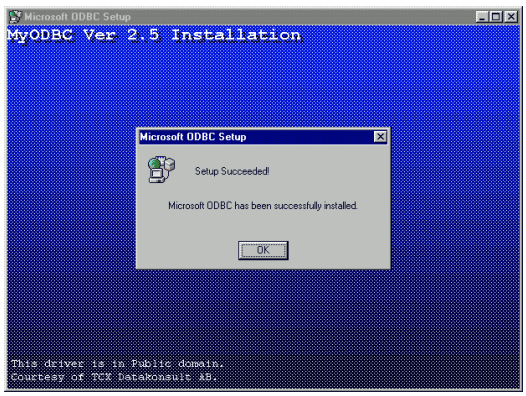
Aby sme mohli v aplikáciách používať ODBC driver, musíme ho najprv naištalovať. Vyberieme si ten typ, ktorý je určený pre náš operačný systém. Pre Windows 95/98 je to jeden typ, pre Windows NT zase druhý typ. Obidva si môžete stiahnúť z mojej web stránky www.mior.host .sk. Princíp inštalácie je veľmi podobný a prípadné rozdiely skúsený užívateľ Windows NT vyrieši sám, preto sa budem venovať iba verzii pre Windows 95/98.
A aby sme to mali akési ucelenejšie, poďme na to spolu metódou postupných krokov:

1) Po stiahnutí príslušného drivera MyODBC v zazipovanom tvare tento rozbalíme do pomocného adresára
2) V tomto adresári spustíme program ***SETUP***
3) Na obrazovke sa objaví inštalátor ODBC. Nech vás neprekvapí,že je to Microsoft ODBC Setup utilita, budeme inštalovať MySQL ODBC driver verzie 2.5, tak ako to vidíme na obrázku č.16-1:

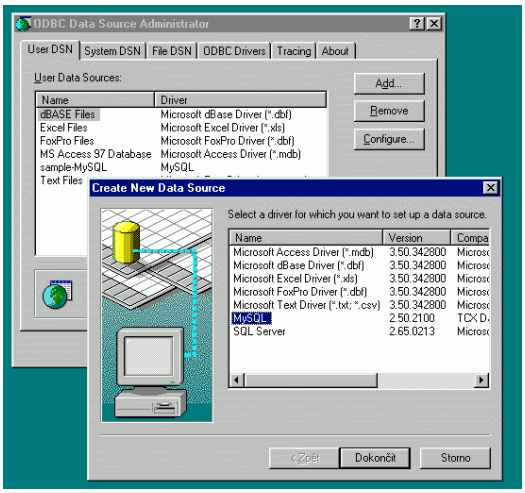


4) Klikneme na **Continue** a v okne **Install drives** kliknutím zvýrazníme položku **MySQL** (Ak tak neurobíme, driver pre MySQL sa nenainštaluje a my budeme hľadať chybu). Pozri obr.č.16-2:

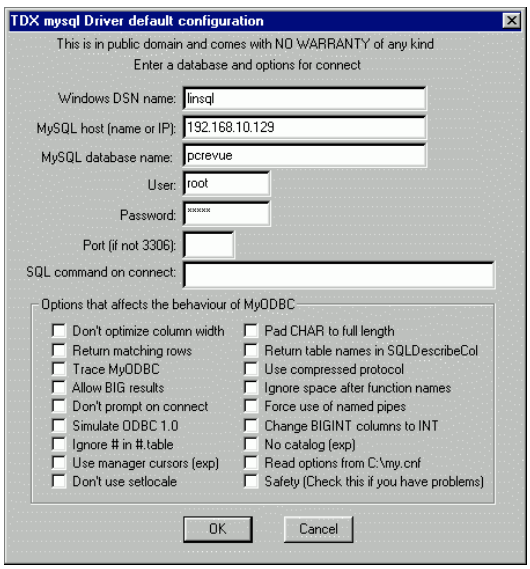
5) Po prebehnutí inštalácie sa obrazí na obrazovke obr. č. 16-3. Týmto bodom sa ukončí pomerne nudná časť inštalácie ODBC driveru (= drajveru alebo ovládača. Sú to tri rôzne slová s rovnakým významom.)



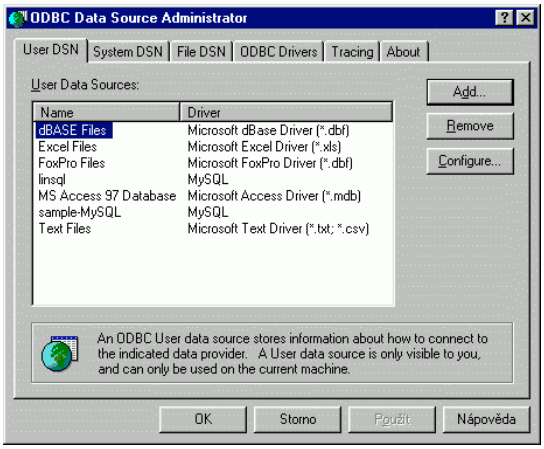
6) Po tomto môžeme pomocný adresár zmazať, aby zbytočne nezaberal miesto a samotný zazipovaný súbor MyOBDC si odložíme (jeden nikdy nevie...).
7) Teraz musíme vykonať nastavenie ODBC drivera. To urobíme takto:
Prejdeme do menu **Štart - Nastavenia - Ovládacie panely** a pohľadáme ikonku s názvom **32bit ODBC**. Klikneme na túto ikonku a zobrazí sa okno **ODBC Data Source Administrator** . Na záložke s názvom **User DSN** vidíme názvy **User Data Sources** (užívateľské dátové zdroje). Ak prebehla inštalácia MyODBC driveru v poriadku, mali by sme tu uvidieť dátový zdroj s názvom ***sample-MySql*** s driverom MySQL. Toto je príklad užívateľského dátového zdroja, ktorý sa vytvoril pri inštalácii MyODBC drivera. Ak stlačíme tlačítko **Configure...**, môžeme si prezrieť jeho nastavenia. Doporučujem tento príklad ponechať ako vzor. My si vytvoríme vlastný užívateľský dátový zdroj. Preto klikneme na tlačítko **Add** a zobrazí sa ďaľšie okno s názvom **Create New Data Source** (obr.č. 16-4):



8) V okne sa zobrazí zoznam niekoľkých driverov, hlavne od firmy Microsoft, ktoré už boli nainštalované súčasne s inštaláciou operačného systému, ako napr. Microsoft Access Driver, Microsoft dBase Driver a podobne. Zároveň sa tam nachádza nami požadovaný MySQL driver. Presunieme naň kurzor myši a kliknutím ho zvýrazníme, čím sme selektovali, že sa ďalej budeme zaoberať práve týmto driverom. Klikneme na **Dokončit** a objaví sa konfiguračný formulár mysql drivera – obr.č. 16-6:



Vyplníme jednotlivé položky takto (toto je moja inštalácia, vy si ju prispôsobte podľa seba):
Prvý riadok je **Windows DSN name** – zvolíme ľubovoľné meno, napr. ***linsql***. V druhom riadku vyplníme meno alebo IP adresu počítača, na ktorom beží MySQL server. Ak beží server na tom istom počítači, kde je aj spustená klientská aplikácia, zadáme *localhost*, ak beží server inde, zadáme meno alebo IP adresu konkrétneho počítača. (Mne beží MySQL server na linuxe a Excel na inom stroji, a nepoužívam DNS – nemýliť s **DSN**!!!- preto som zadal IP adresu linuxu). V treťom riadku **MySQL database name** zadáme názov databáze, z ktorej budeme čerpať dáta. Jej meno si zvolíme podľa našej potreby, napr. ***pcrevue***. Zostáva vyplniť meno užívateľa **User**, čo musí byť meno, ktoré MySQL databáza pozná a má príslušné prístupové práva. V položke **Password** zadáme heslo zadaného užívateľa. Vidíte, že je skryté pod hviezdičkami, aby ho nikto nemohol odpozorovať. Ostatné položky nevyplňujeme, ak sme ovšem nespustili MySQL server v špeciálnom režime. Naše nastavenia potvrdíme skliknutím na tlačítko **OK**. Zobrazí sa okno **ODBC Data Source Administrator** a vidíme, že pribudol nami definovaný užívateľský zdroj s názvom ***linsql***, pracujúci s driverom MySQL – obr.č.16-6:



Tým sme ukončili inštaláciu a nastavenie ODBC drivera pre MySQL server.

Vytvorenie vzorovej tabuľky

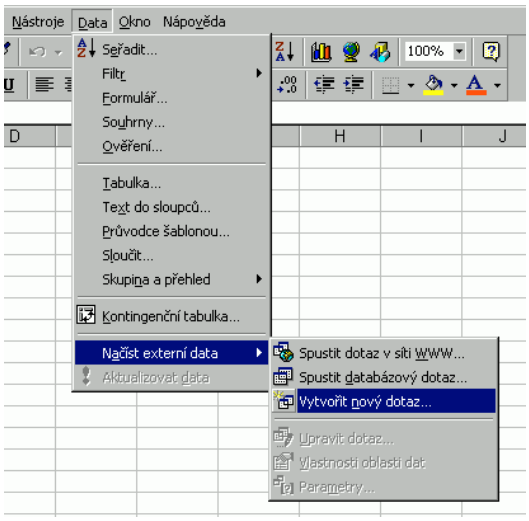
Aby sme si mohli ukázať činnosť ODBC drivera v praxi, vytvoríme si vzorovú databázu a tabuľku. Začneme vytvorením databáze, ktorú pomenujeme rovnako, ako sme jej meno zadefinovali v nastavení drivera, teda ***pcrevue***. (Samozrejme si môžete meno zmeniť). Potom v tejto databáze vytvoríme tabuľku s ľubovoľným názvom, napr. ***tabulka1***. Bude obsahovať údaje o predaji súčiastok za rok 2000, uložené v dvoch stĺpcoch **mesiac** a **pocet**. Jej obsah by mohol vyzerať takto:

mesiac	pocet
január	159
február	256
marec	456
apríl	423
máj	359
jún	223
júl	156
august	623
september	265
október	121
november	226
december	452

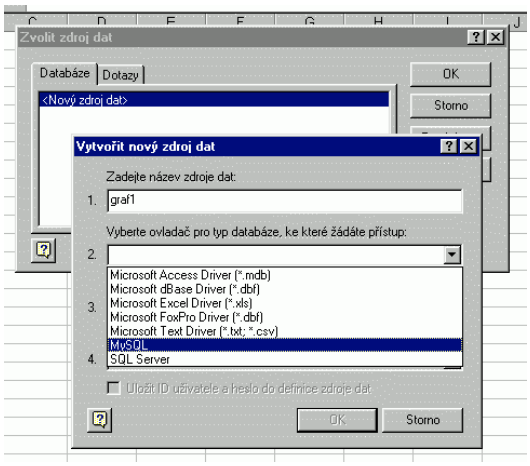
Ako sa taká databáza a tabuľka vytvorí, to už samozrejme dobre vieme, ale ak sa vám nechce ju ručne napĺňať, môžete použiť skript, ktorý nájdete na mojej web stránke.

Klient

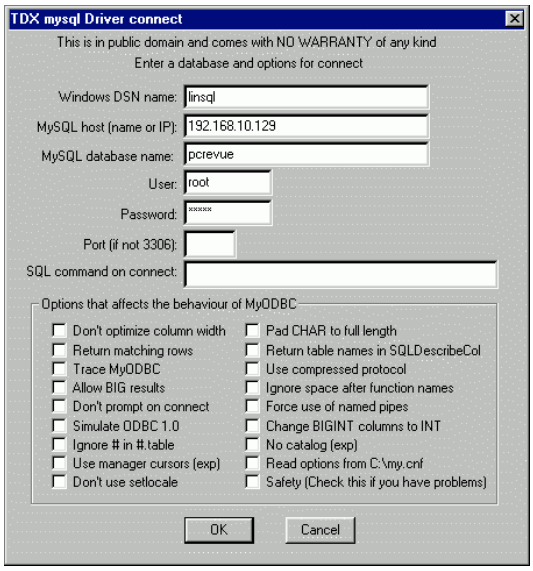
Ako sme si už povedali, klientom, komunikujúcim s MySQL serverom cez MyODBC driver môže byť ľubovoľná aplikácia, schopná používať klasické SQL príkazy. Takou bude v našej dnešnej ukážke aj MS Excel. Ten vie pomocou MS Query volať príslušný ODBC driver a poslať mu SQL príkaz. Výsledok príkazu dokáže spracovať a zobraziť vo svojom liste.
Spusťme MS Excel. Kliknime na menu **Dáta – Načítať externé dáta – Vytvoriť nový dopyt** (obr.č.16-7):



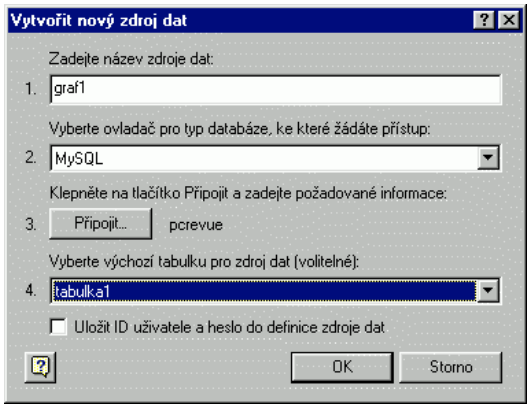
Otvorí sa okno **Zvoliť zdroj dát**. Kliknutím zvýrazníme položku **<Nový zdroj dát>** a stlačíme **OK**. Objaví sa nové okno **Vytvoriť nový zdroj dát** – obr.č.16-8:



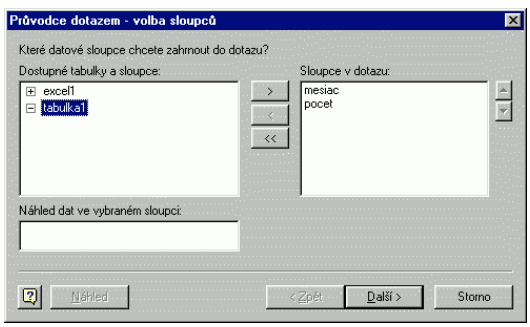
Začneme vypĺňať jednotlivé riadky.
Do prvého zadáme ľubovolný názov zdroja dát, napr. **graf1**. V druhom riadku klikneme na rozbaľovaciu šípku, kde sa rozbalí roletové menu s ovládačmi – drivermi pre jednotlivé databáze. Zo zoznamu vyberieme typ **MySQL**. Vrátime sa do prvého okna a na treťom riadku stlačíme tlačítko **Pripojiť**. Zobrazí sa už známy formulár tentokrát s názvom **TDX mysql Driver Connect**, ktorý vyplníme úplne presne, ako v predchádzajúcom formulári. Ak nedodržíme presnosť vyplnenia, spojenie sa neuskutoční!!! (obr.č.16-9):



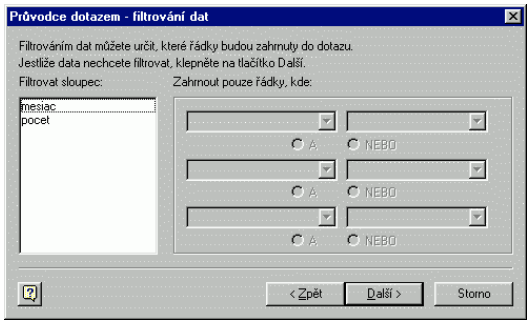
Po stlačení tlačítka **OK** sa vrátime do okna **Vytvoriť nový zdroj** – obr.č.16-10:



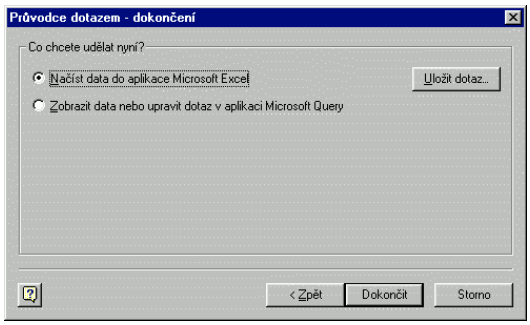
Ak sa zobrazí okno s chybovou správou, niekde sme sa dopustili chyby, najpravdepodobnejšie sme nevyplnili obidva formuláre ROVNAKO!
Ak však prebehne všetko v poriadku, vedľa tlačítka **Pripojiť** sa objaví názov databáze, do ktorej sme sa zadaním vo formulári pripojili, teda v tomto prípade ***pcrevue***. Zároveň sa vo štvrtom riadku ponúknu názvy tabuliek, uvedených v danej databázi. Stlačíme tlačítko **OK**.
Objaví sa okno **Sprievodca dopytom – voľba stĺpcov** – obr.č.16-11:



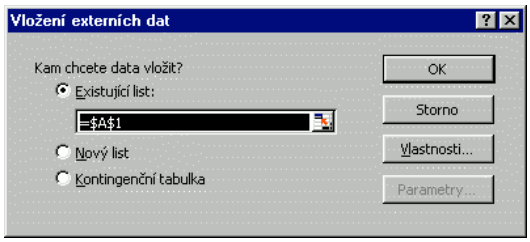
V tomto okne zvolíme príslušnú tabuľku, vyberieme požadované stĺpce tabuľky (v našom prípade úplne všetky dva) a stlačíme tlačítko **Ďalší>.**
Objaví sa okno **Sprievodca dopytom – filtrovanie dát** (obr.č.16-12):



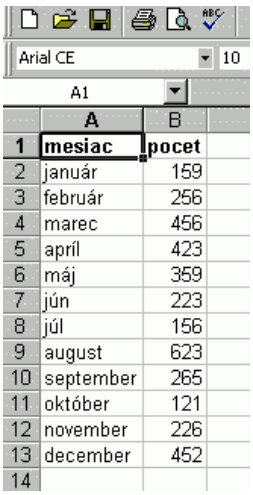
V tomto okne by sme mohli nastavovať rôzne filtre, aby sme získali dáta, ktoré skutočne potrebujeme. Pravda, v našom ilustračnom príklade nepoužijeme žiadny filter a preto stlačíme **Ďalší**>.
Objaví sa nové okno **Sprievodca dopytom – dokončenie** (obr.č.16-13):



Tu sa ponúka možnosť, či rovno načítať dáta do aplikácie MS Excel, alebo použiť MS Query na ďalšiu úpravu dát. My si vyberieme prvú možnosť a stačíme tlačítko **Dokončiť**.
Objaví si ďalšie okno **Vloženie externých dát** – obr.č.16-14:



Vyberieme si, kam sa majú nami žiadané dáta uložiť. Zase zvolíme prvú možnosť – uloženie do existujúceho listu. Môžeme vybrať bunku ľavého horného rohu, kam sa dáta načítajú. Stlačíme **OK** a dostaneme sa do listu Excelu a dáta sú na svojom mieste – obr.č.16-15:



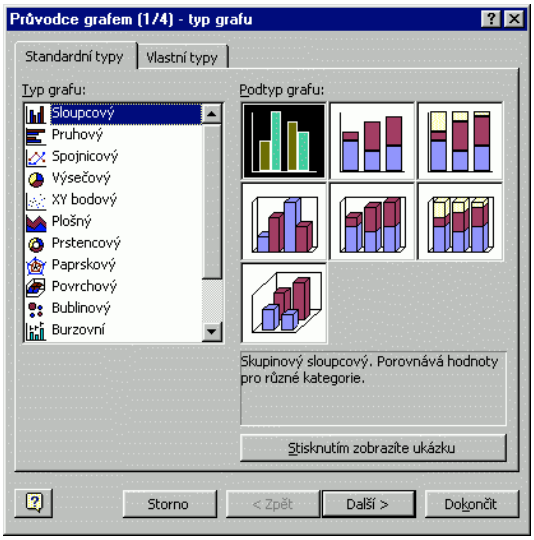
Arial CE 10

A1

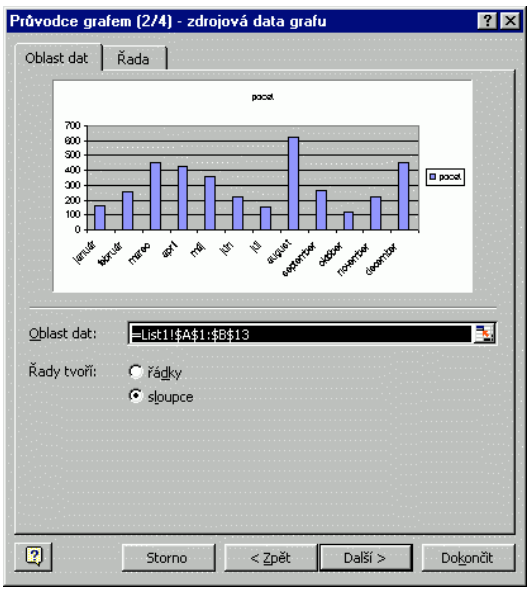
	A	B
1	mesiac	pocet
2	január	159
3	február	256
4	marec	456
5	apríl	423
6	máj	359
7	jún	223
8	júl	156
9	august	623
10	september	265
11	október	121
12	november	226
13	december	452
14		

No čo, dobré, nie? Ale poznáte šéfov. Ich holé čísla nezaujímajú. A tak mu pripravíme graf!

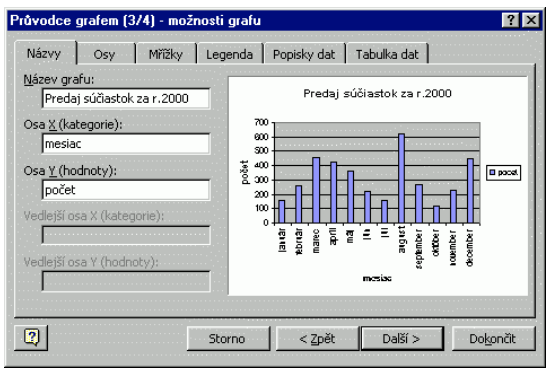
Na to použijeme **Sprievodcu grafom**, čo je malá ikonka napravo. Klikneme na ňu a zobrazí sa prvé okno **Sprievodcu grafom – typ grafu** (obr.č.16-16):



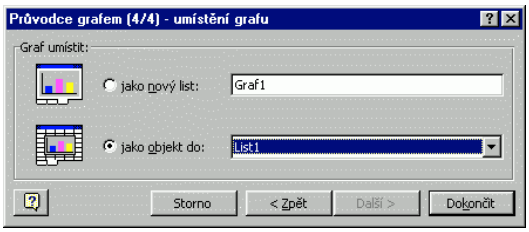
Tu si môžeme vybrať graf podľa vlastnej chuti. Mne sa páčil hned prvý – stĺpový, ale vy sa vyhrajte sami.
Stlačíme **Ďalší>**.
Objaví sa druhé okno sprievodcu – **zdrojové dáta grafu** (obr.č.16-17):



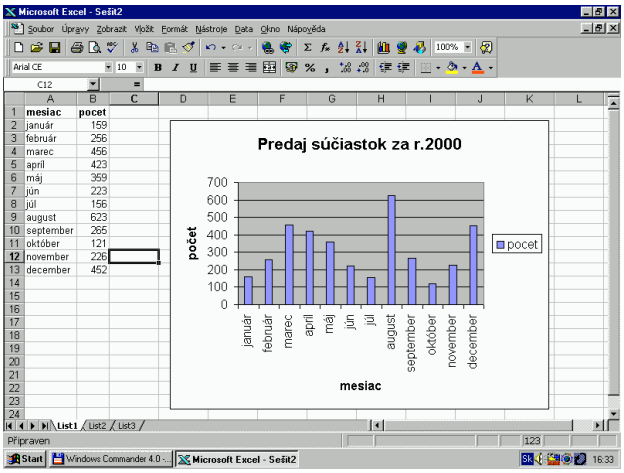
Tu máme opäť niekoľko možností úpravy grafu, ale my stlačíme zase tlačítko **Ďalší>**.
Objaví sa tretie okno sprievodcu – **možnosti grafu** (obr.č.16-18):



Tu sa trocha pohráme. Pomenujeme graf, osu X a osu Y. Stlačíme **Ďalší>.**
Objaví sa posledné okno sprievodcu grafom – **umiestnenie grafu** (obr.č.16-19):



Vyberieme si umiestnenie ako objekt do **List1** a stlačíme **Dokončiť**.
Vrátime sa do listu aplikácie a vidíme, že okrem tabuľky máme na liste aj nádherné grafické vyjadrenie dát – obr.č.16-20:



A to bolo cieľom dnešného snaženia. Ak vás napadajú rôzne iné vylepšenia a aplikácie, ste na správnej ceste moderného tvorcu prezentácie dát.

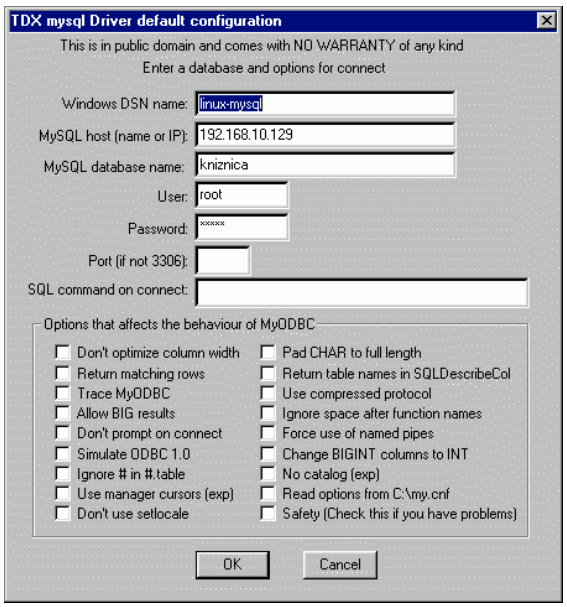
Možno vás napadne, že Excel je len taký jednosmerný prostriedok. Dokáže dáta zobrazovať, ale už ich nevie modifikovať alebo dopĺňať. Ozaj by to bolo efektné, aby sme z akejsi oknoidnej aplikácie dáta aj menili, dopĺňali alebo mazali. A to si vysvetlíme nabudúce.

Malé veľké databázy II / 7.časť

V minulej časti sme si ukazovali pripojenie na MySQL server pomocou ODBC. Ako klienta sme použili pomerne známy program MS Excel, kde sa zobrazila tabuľka a k nej sme urobili graf. Toto sa hodí na prezentáciu dát, ktoré sú už vytvorené, lebo Excel nedovoľuje dáta, uložené na MySQL serveri, modifikovať. Na túto prácu sú podstatne vhodnejšie vyššie programovacie jazyky ako Pascal, Delphi, C, C++ a iné.
V týchto jazykoch je možné vytvoriť aplikáciu, ktorá bude k dátam pristupovať dvomi spôsobmi – buď cez ODBC, alebo cez API pomocou konkrétnych komponent v tom-ktorom jazyku.

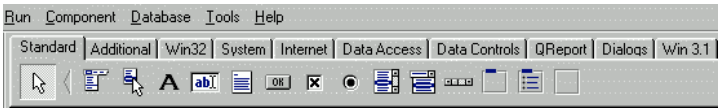
Výhody a nevýhody ODBC
Použitie spojenia cez ODBC má niekoľko výhod a nevýhod:
Výhodou spojenia cez ODBC je, že vytvorený projekt môžeme bez úprav použiť na hociktorý SQL server. Stačí použiť iba správny ODBC driver k žiadanému SQL serveru. Teda ak vytvoríme aplikáciu, ktorá bude pristupovať k serveru MySQL (pomocou MyODBC) a po určitom čase budeme chcieť prejsť na iný SQL server, napr. PosgreSQL, Informix alebo Oracle, stačí, ak použijeme príslušný ODBC driver a aplikáciu nie je potrebné znova vytvoriť. Náš užívateľ vlastne ani nevie (a asi ho to ani nezaujíma), na ktorý SQL server pristupuje, on (alebo ona) potrebuje účtovať, tlačiť doklady, tvoriť výstupné zostavy a podobne. Aplikácia je z jeho pohľadu úplne rovnaká, zvykol si na ňu, ba dokonca, ak mu vyhovuje, nemá vôbec rád akékoľvek zmeny (vlastná skúsenosť). No a my, my máme zjednodušenú prácu.
Asi jedinou nevýhodou použitia ODBC je určité spomalenie prístupu k dátam oproti použitiu prístupu cez API. Je to vlastne logické. Aplikácia pošle SQL dopyt príslušnému ODBC, ten to musí pretransformovať na jazyk, zodpovedajúci konkrétnemu SQL serveru. Ten príkaz vykoná, pošle odpoveď späť ODBC, ten ju transformuje do podoby pre klientskú aplikáciu. Nemusíme sa však báť. To spomalenie nie je pri menej rozsiahlych databázach obmedzujúce a ešte stále je niekoľko možností, ako to „vyladiť“!
Dnes si ukážeme, ako vytvoríme aplikáciu v Delphi od Borlandu, kde použijeme spojenie cez ODBC.

Predpokladajme, že máme nainštalované MyODBC tak, ako sme to robili v predchádzajúcich častiach. Teraz si vytvoríme cvičné spojenie mysql drivera s názvom **linux-mysql**. (možeme si ho, samozrejme pomenovať inak). Prejdeme do *Ovládacích panelov*, klikneme na ikonku *32bit ODBC*, klikneme na tlačítko *Add...* , vyberieme driver *MYSQL* a pokračujeme tlačítkom *Dokončiť*. Zobrazí sa okno ***TDX mysql Driver default configuration*** a vyplníme ho obdobne ako je na obr. 17-1:



Ak máme MySQL server spustený na tom istom počítači, kde bude bežať aj naša klientská aplikácia, vyplníme do položky ***MySQL host*** slovo localhost. (Ja používam MySQL server na Linuxe, kde IP adresa počítača je 192.168.10.129, a klientskú aplikáciu budem tvoriť na počítači s Windows, ktorý ma IP adresu 192.168.10.130.). Toto spojenie využijeme pri našom dnešnom demonštračnom príklade.

Ak chceme pracovať s Delphi, musíme mať vhodnú verziu. To je taká verzia, ktorá na lište komponent obsahuje záložky ***Data Access*** a ***Data Controls*** (obr.č.17-2):



Niektoré verzie, napr. Borland Dephi 6 Personal alebo Kylix Open Edition (ktoré sú zdarma) tieto komponenty nemajú.

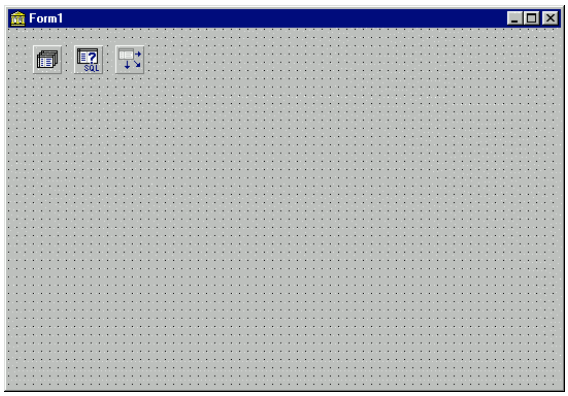
Tak a poďme na to postupne:

Demonštračný príklad
Vytvorme aplikáciu v Delphi, ktorá sa spojí pomocou ODBC k serveru MySQL k databáze **KNIZNICA** a bude zobrazovať dáta z tabuľky **kniha** v okne na formulári. Chceme, aby bol prístup autentizovaný (teda iba po zadaní správneho mena a hesla) a aby sa jednotlivými položkami – záznamami dalo listovať.
Stačí, nič viac (zatiaľ!).

Predpokladom sú vaše aspoň základné znalosti o Delphi a o práci v tomto programovacom jazyku a jeho vývojovom prostredí.

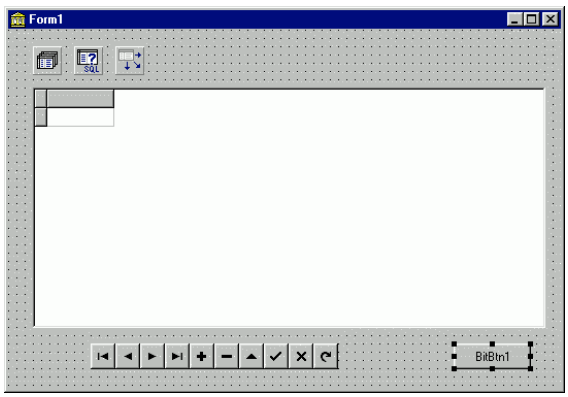
Tvorba základného formulára
Spustíme Delphi. Na jeho formulár položíme tieto komponenty zo záložky *Data Access*:
- **Database1** typu *TDatabase*
- **Query1** typu *TQuery*
- **DataSource1** typu *TDataSource*

tak, ako je to na obrázku č.17-3:

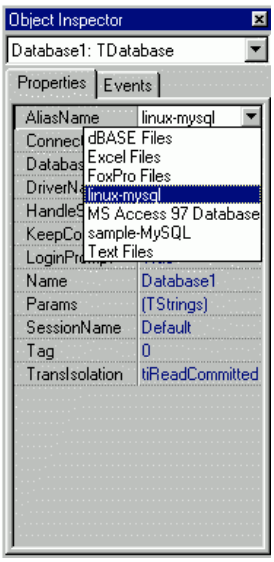


Zo záložky *Data Controls* na formulár doplníme komponenty:
- **DBGrid1** typu *TDBGrid*
- **DBNavigator1** typu *TDBNavigator*

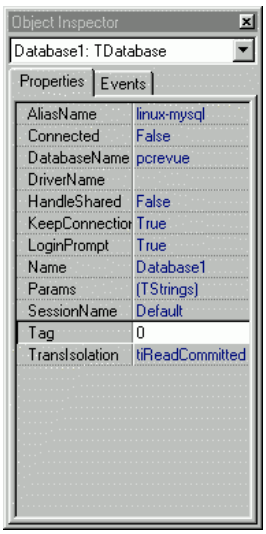
a zo záložky *Additional* pridáme komponentu **BitBtn1** typu *TBitBtn*, tak ako je to na obrázku č.17-4:



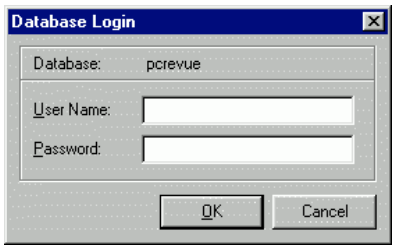
Popis jednotlivých komponent
Database1 typu *TDatabase* poskytuje diskrétny prístup cez spojenie k jednoduchej databázi v databázovej aplikácii. Klikneme na túto komponentu vo formulári a v *Object Inspectore* v záložke ***Properties*** (= vlastnosti) sa presunieme na prvú vlastnosť ***AliasName***. Klikneme na malú šípočku vpravo, čím rozbalíme roletkové menu ponúkaných ODBC spojení (obr.č.17-5):



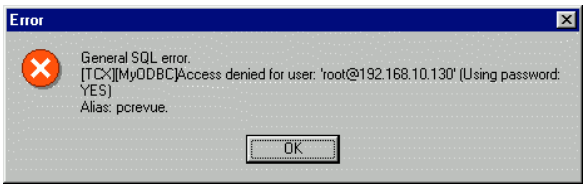
Vyberieme nami vytvorené spojenie **linux-mysql**. (Teda to, ktoré sme vytvorili vyššie popísaným spôsobom). Vyplníme položku ***DatabaseName***, ktorú nazveme napr. **pcrevue** (obr.č.17-6):



Teraz klikneme na vlastnosť ***Connected*** a nastavíme ju na hodnotu ***True***. Objaví sa prihlasovacie okno ako na obrázku č.17-7:



Zadáme príslušné meno, heslo a potvrdíme tlačítkom **OK**. Ak sme zadali správne údaje, teda správne meno a správne heslo, ktoré SQL server pozná a na základe ktorých umožní prístup, prihlasovacie okno zhasne bez vyhlásenia chyby. To je dôkazom, že sa podarilo nadviazať spojenie s SQL serverom. Ak sme zadali niektorú položku chybne, dostaneme chybovú hlášku. Príklad takejto hlášky je na obr. č.17-8:



Z hlášky vidíme, kde sa asi stala chyba – máme zakázaný prístup. Treba chybu vyhľadať a odstrániť.

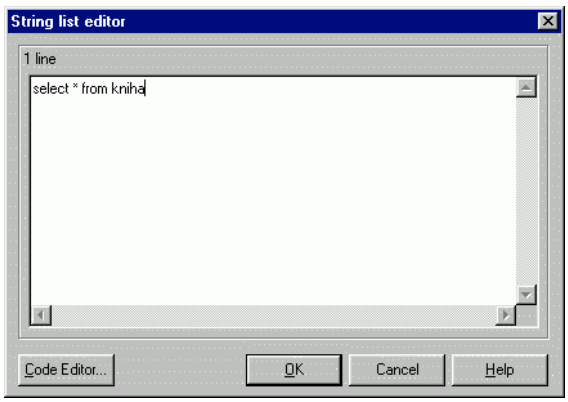
(Poznámka: tu sa robia tieto najčastejšie chyby: buď bolo zadané nesprávne meno alebo heslo, alebo na serveri MySQL nemá prihlasujúci sa užívateľ definované prístupové práva, alebo je nesprávne vyplnená karta driveru ODBC z obr.č.17-1).

Delphi má jednu výbornú vlastnosť. Už počas tvorby aplikácie (teda ešte pred samotnou kompiláciou!) v IDE prostredí umožňuje niektoré dôležité činnosti, ako vytvorenie spojenia do databáze, prístup k tabuľkám, zobrazenie dát a podobne. Túto fantastickú vlastnosť využijeme, čo si popíšeme neskôr. Takto môžeme odhaliť chyby skôr, ako dopracujeme celý projekt.
Ak sa teda spojenie podarilo, v položke ***Connected*** *Object Inspectora* sa zobrazí hodnota ***True***.

Teraz klikneme na komponentu **Query1**.
Query1 typu *TQuery* vykonáva query (= dopyt) na pripojené dáta.
V *Object Inspectore* vlastností **Query1** klikneme na šípočku položky (vlastnosti) ***DatabaseName*** a vyberieme z roletkového menu položku ***pcrevue***.
Povedali sme si, že komponenta Query vykonáva dopyty, teda SQL príkazy k danému SQL serveru. Preto klikneme v *Object Inspectore* na vlastnosť ***SQL***. Zobrazí sa okno ***String list editor***. V tomto okne zadáme požadovaný SQL príkaz. Ale aký to bude?
Povedali sme si, že chceme prezerať záznamy v tabuľke **kniha**. Takže SQL príkaz bude takýto:

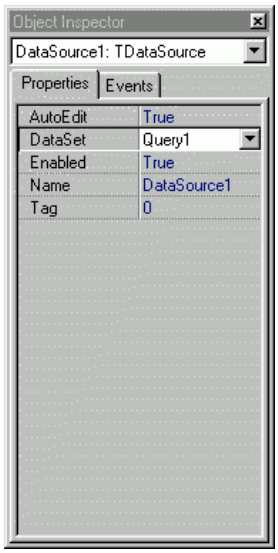
select * from kniha

Poznávame ho? Ale samozrejme, veď je to výpis všetkých záznamov z tabuľky **kniha**!
Tento SQL príkaz napíšeme do okna ***String list editora*** (obr.17-9):

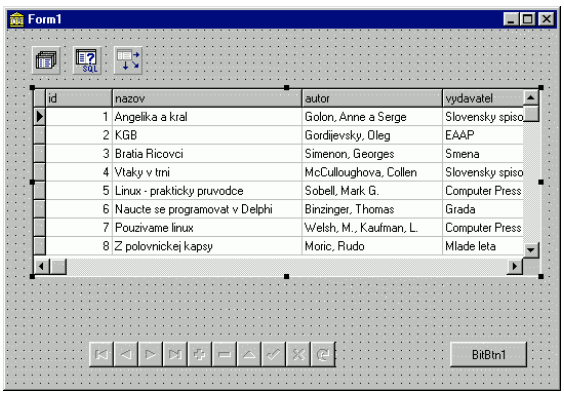


Klikneme na **OK**.
Prejdeme na prvú vlastnosť ***Active*** a obdobným spôsobom zvolíme hodnotu ***True***. Ak Delphi nevyhlási chybu, všetko pokračuje podľa plánu. V prípade chyby ju analyzujeme a odstránime. Skontrolujeme *Object Inspector*, či je všetko zachované tak, ako sme nastavili a prejdeme na komponentu **DataSource1**.

DataSource1 typu *TDataSource* tvorí akýsi interfejs medzi komponetami na vytvorenie spojenia k dátam (*Database*, *Query*) a komponentami na manipuláciu s takto získanými dátami (*DBGrid*, *DBNavigator*).
Inak povedané, **DataSource1** sa naviaže na komponentu **Query1** a na komponentu **DataSource1** sa naviaže komponenta **DBGrid1**.
Klikneme teda na komponentu **DataSource1**. V *Object Inspectore* klikneme na vlastnosť ***DataSet*** a vyberieme (ponúkanú) hodnotu ***Query1*** (obr.č.17-10):



Pristúpime k **DBGrid1**.
DBGrid1 typu *DBGrid* zobrazuje a manipuluje s dátami, získanými z dátového zdroja. Preto vo vlastnostiach **DBGrid1** v položke ***DataSource*** nastavíme hodnotu ***DataSource1***.
Aha, čo sa to zrazu stalo? V mriežke **DBGrid1** sa objavili skutočné, teda ostré dáta z tabuľky **kniha** (obr.č.17-11):

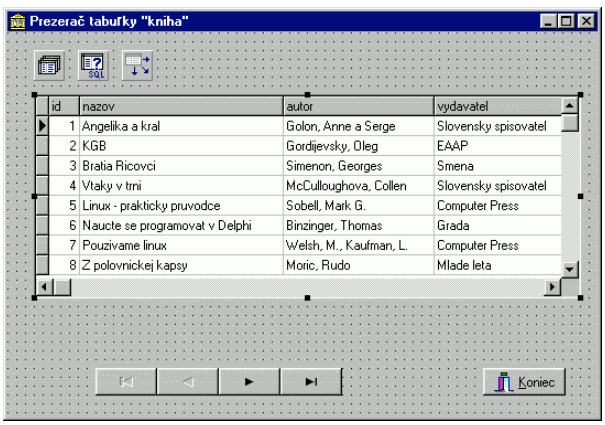


A to je tá výborná vec v Delphi, ktorú som už spomínal!

Aby sme mohli listovať jednotlivými záznamami dopredu, dozadu, na začiatok alebo na koniec, na to použijeme konponentu **BDNavigator1**. Stačí, ak v jej vlastnostiach aktivujeme položku ***DataSource*** a nastavíme hodnotu na ***DataSource1***. Je to identické ako v prípade komponenty **DBGrid**. Nemusíme mať zobrazené všetky tlačítka, ktoré *DBNavigator* ponúka. Stačí, ak vo vlastnosti ***VisibleButtons*** aktivujeme položky ***nbFirst***, ***nbPrior***, ***nbNext*** a ***nbLast***. Už názvy hovoria, aké sú ich funkcie. Všimnime si, ako sa zmenia samotné tlačítka *DBNavigarora*.

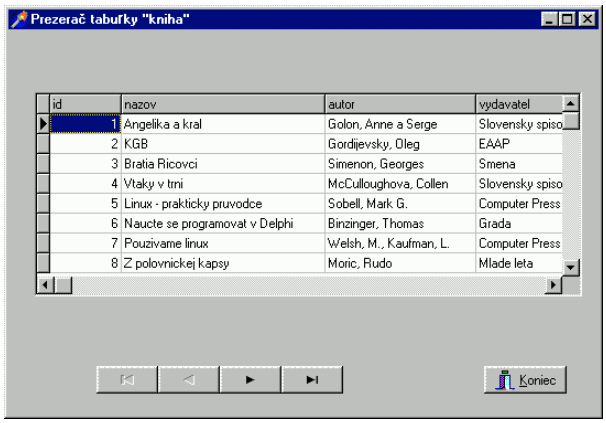
Ostáva posledná komponenta **BitBtn1** typu *TBitBtn*. Je to grafické tlačítko. V podstate už nemá žiadnu funkciu, zviazanú priamo s databázavým strojom. My ju však použijeme ako ukončovacie tlačítko celej našej aplikácie. Viem, viem, budete namietať, že aplikácia sa dá uzavrieť aj kliknutím na krížik v jej pravom hornom rohu, tak ako všetky oknoidné aplikácie. No ale uznajme, tlačítko je efektnejšie.
Takže v jeho vlastnostiach zmeníme tieto položky:
Položku ***Kind*** zmeníme na hodnotu ***bkClose***, ktorá zabezpečí ukončenie celej aplikácie. Položku ***Caption*** zmeníme na ***&Koniec***, čím zmeníme popis tlačítka.

Úplne záverom klikneme na základný formulár **Form1** typu *TForm*, kde upravíme vlastnosť ***Caption***. Napíšeme do nej text – **Prezerač tabuľky "kniha"** – čím zmeníme aj titulok celej aplikácie. Výsledok vidíme na obrázku č.17-12:



Nastala záverečná etapa – kompilácia. Takto vytvorený program zkompilujeme. Ak sme neurobili žiadnu chybu, preklad prebehne v poriadku a vznikne hotová aplikácia ako .exe súbor.

Skompilovanú aplikáciu spustíme. Vidíme, že sa zobrazí vstupné dialógové okno, ktoré požaduje zadanie mena a hesla. Po správnom zadaní sa zobrazí hlavná obrazovka, tak ako ju vidíme na obr.č.17-13:



Všimnime si, že na skompilovanej aplikácii nie je vidieť tzv. neviditeľné (non-visible) komponenty, ako sú *DataSource*, *Query* alebo *Database*. Veď to ani nie je potrebné, aby ich bolo vidieť, stačí, ak sme nastavili ich vlastnosti podľa našich potrieb.
Môžeme byť právom na seba hrdí. Urobili sme funkčnú aplikáciu, ktorá je prenositeľná na všetky počítače so systémom WindowsXX. Stačí, ak ju tam nakopírujeme, nainštaľujeme MyODBC a správne nastavíme. A len tak, medzi nami, toto sa dá krásne zautomatizovať, čo zruční delfisti vedia, ale to nie je cieľom dnešnej lekcie.

Už počujem vaše hlasy – *„My ale nechceme len prezerať! My chceme pridávať, editovať, mazať! Len tak je aplikácia to pravé orechové!“*.
Máte pravdu. Dnešná časť bola iba ukážková, veľmi jednoduchá (aj keď ani jej nemožno uprieť určitú funkčnosť!).
A tak ako sa robia ostatné „kúzla“ s dátami si povieme nabudúce.
A na okraj, zdrojové texty tohoto príkladu si môžete stiahnúť v mojej web stránky www.mior.host.sk.

Malé veľké databázy II / 8.časť

Keďže sa aj v dnešnej časti budeme venovať skôr „delfistom“, tak pre ostatných „maj-es-que-el-ákov“ aspoň jedna dobrá správa:
Niekoľko testerov vykonalo porovnania MySQL verzie 4 beta s verziou 3.23.xy. Keďže sa priznali, že zrovna nefandia MySQL, ich hodnotenie považujem za nenadnesené a tak priam môžem povedať, že ma ich výsledky nadchli.
Takže: MySQL ver. 4 beta je zatiaľ pri malom počte záznamov asi dvakrát pomalšia ako verzia 3.23.xy. Rýchlosť sa prejaví až pri veľmi veľkých počtoch záznamov. Dá sa však predpokladať, že sa toto podarí doriešiť do vypustenia ostrej verzie. Veľmi potešujúce je, že sub-selekty (vnorené selekty = select (select.....)) fungujú veľmi dobre a teraz to najdôležitejšie: Transakcie, ktoré doteraz MySQL nepodporovala, a ktoré jej mnohí odporcovia vytýkali a v tejto verzii sú už implementované, pracujú veľmi spoľahlivo! To skutočne povyšuje MySQL na úroveň veľkých databáz, kde sú pomerne citlivé dáta (hlavne finančné operácie, zásoby, sklady , elektronický obchod a pod.).
My ako skúsení databázisti už nájdeme pre MySQL 4.x to správne uplatnenie! Už sa teším na ostrú verziu...

Vráťme sa však naspäť k nášnu projektu. V ňom dokážeme veľmi pekne prezerať dáta v určitej tabuľke, konkrétne **KNIHA**, ale nevedeli sme ich zaťiaľ doplniť alebo zmeniť.
Dnes si upravíme aplikáciu z minulej časti seriálu tak, že ju rozšírime o možnosť pridávať záznamy.

Zjednodušenie obsahu tabuľky
Už počiatkom našej práce s MySQL sme vytvorili tabuľku **KNIHA**, ktorá má túto štruktúru:

```
CREATE TABLE `kniha` (
  `id` int(11) NOT NULL auto_increment,
  `nazov` varchar(40),
  `autor` varchar(30),
  `vydavatel` varchar(25),
  `cis_odd` int(11),
  `cena` decimal(5,2),
  `poznamka` varchar(25),
  PRIMARY KEY  (`id`)
)
```

Dnes, pre ilustráciu budeme využívať iba tieto stĺpce:
- ***id***
- ***nazov***
- ***autor***
- ***vydavatel***
- ***cena***

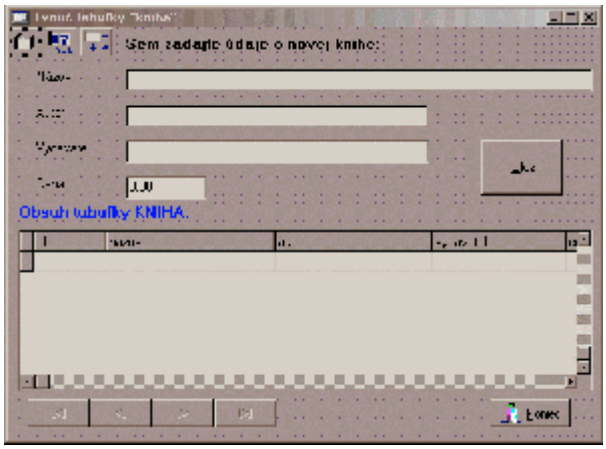
Ostatné stĺpce zostanú nevyplnené.

Vieme, že stĺpec *id* má autoinkrementačnú vlastnosť, takže ho nemusíme zadávať, stačí, ak v príkaze *INSERT* necháme prázdne miesto, SQL server doplní číslo sám. (Spomeňme si na to, keď sme preberali autoinkrementáciu!).

Aby sme mohli dopĺňať ostatné 4 položky, musíme v našej aplikácii použiť editačné polia.

Úprava formulára
Komponentu *Form1* typu *TForm* upravíme tak, že doplníme tieto komponenty:
- *Edit1, Edit2, Edit3, Edit4* typu *TEdit*, ktorá umožňuje zadávanie textu do prázdnej kolónky, s ktorými budeme ďalej pracovať. Tieto komponenty nájdeme na záložke *Standard*
- *Button1* typu *TButton* zo záložky *Standard*, ktorú budeme používať na zápis editovaných dát do tabuľky na SQL serveri.
- *Label1, Label2, Label3, Label4, Label5 a Label6* typu *TLabel* zo záložky *Standard*. Tie použijeme na popis jednotlivých editačných polí.

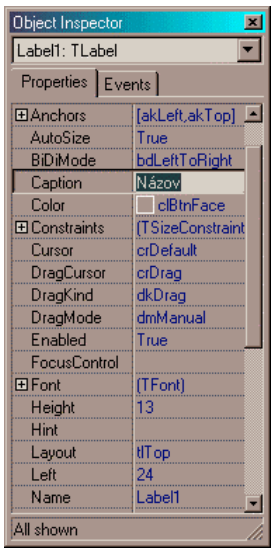
Zároveň upravíme formulár tak, aby sme aspoň trochu ergonomicky rozmiestnili všetky vizuálne komponenty nového projektu. Jedno z možných riešení je na obr.č.18-1:



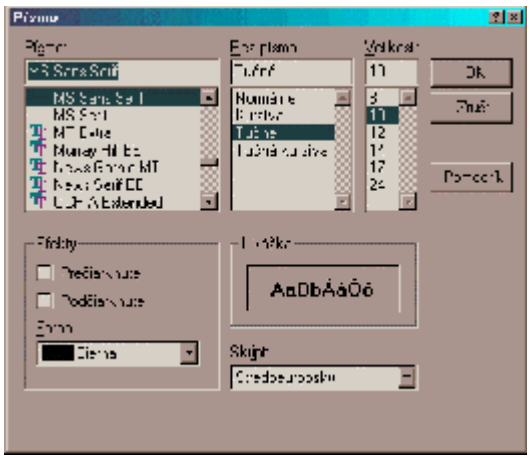
Úprava jednotlivých komponent
Teraz pristúpime k úprave jednotlivých komponent:
Editačné polia *Edit1*, *Edit2* a *Edit3* uravíme v Object Inspectore tak, že vlastnosť **Text** zostane prázdna (obr.č.18-2):

Komponentu *Edit4* upravíme tak, že do vlastnosti **Text** v Object Inspectore vpíšeme text „***0.00***“. Tým naznačíme užívateľovi, že v tomto okne sa očakáva zadávanie peňažnej hodnoty.

Pristúpime k úprave komponenty *Label1*. V Object Inspectore vo vlastnosti **Caption** napíšeme text „***Názov***“ (obr.č.18-3).



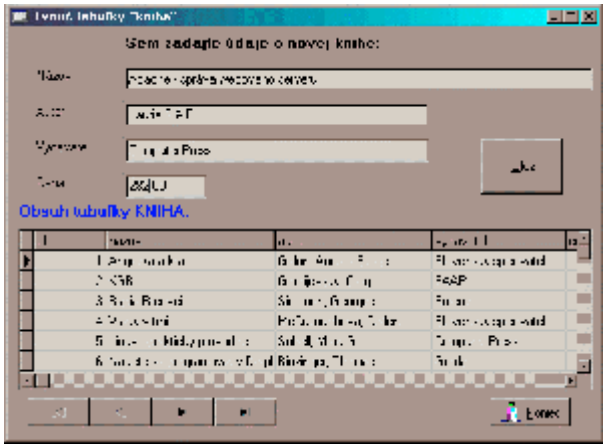
Všimnime si, že sa takto zmenila aj komponenta. Podobým spôsobom upravíme aj ostané komponenty *Label2*, *Label3* a *Label4*, ktorým **Caption** zmeníme na „***Autor***“, „***Vydavateľ***“ a „***Cena***“.
Label5 použijeme ako nadpis „***Sem zadajte údaje o novej knihe***:“, a upravíme ho obdobným spôsobom. Aby sme dosiahli zmenu písma, klikneme v Object Inspectore najprv na vlastnost **+Font**. Tu sa objaví malé tlačítko s tromi bodkami, na ktoré opätovne klikneme. Vtedy sa zobrazí štandartné windowsovské okno na úpravu písma (obr.č.18-4):



Obdobným spôsobom upravíme komponentu *Label6*, ktorá vytvorí text „**Obsah tabuľky KNIHA:**“
Na záver upravíme komponentu *Button1*. Jej úprava je veľmi podobná úprave komponent *Label*. Stačí, ak v okne Object Inspectora zmeníme vlastnosť **Caption** na **„&Ulož**“. Ako zdatní delfisti vieme, že znak „&“ - ampersand znamená, že za ním následujúce písmeno bude tzv. horúca klávesa. V našom prípade kombinácia *Alt-U* spôsobí to isté, ako kliknutie myšou na tlačítko **Ulož**. Len pre úplnosť - horúce klávesy sa poznajú podľa podtržítka pod písmenom (napr. **U**).

Vizuálnu úpravu by sme mali za sebou. Tá závisí na estetickom cítení programátora, prípadne požiadaviek budúceho užívateľa.

Podstatne dôležitejšia je funkčná úprava.
Teraz nastavíme v komponente *Database1* vlastnosť **Connected** na ***True*** a v komponente *Query1* vlastnosť **Active** na ***True***. Vyplníme správne prihlasovacie okno a uvidíme v mriežke skutočné dáta. (Toto sme si vysvetlili minule).
Ak teraz skompilujeme náš projekt, uvidíme viac vizuálnych komponent. Môžeme skúsiť vyplniť editačné polia, podobne ako na obrázku č.18-5:



Ak teraz stlačíme tlačítko **Ulož**, nič sa nevykoná a jeho funkčnosť ostáva nezmenená, teda rovnaká ako v minulej časti seriálu. Prečo?

Nestačí do projektu pridať pár komponent, musíme ešte zabezpečiť, aby sa po stlačení tlačítka vykonala určitá činnosť.

Úprava činnosti tlačítka Ulož
Keď v Borland Delphi klikneme na určitú komponentu, otvorí sa okno zdrojového kódu projektu, napr. ***unit1.pas***.
Automaticky sa zabezpečí, že sa pridá nová procedúra. Procedúra má meno formulára a komponenty v ňom, na ktorú sme klikli. V našom prípade sa objaví okno s touto procedúrou:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Telo procedúry ostáva voľné. A práve v tejto chvíli nastáva úloha pre programátora, aby sem vpísal zdrojový kód, čo má tlačítko po jeho stlačení vykonať.
(A tu je to miesto, kde už funkcie negeneruje Delphi, ale ich vytvára človek. Podľa toho, aký kód napíše, sa hodnotí kvalita výsledného produktu. Treba mať na pamäti, že existuje mnoho ciest, ktoré vedú k rovnakému cieľu a o žiadnej rutine (= funkčná čiastočka kódu) nemožno povedať, že je jedine správna a najlepšia.)

Aby sme mohli napísať funkčnú rutinu, musíme si najprv rozmyslieť, čo má také kliknutie na tlačítko, a teda konkrétna procedúra vykonať.

Poďme sa na to pozrieť najprv logicky:
a) na tlačíko **Ulož** klikáme až po vyplnení editačných polí
b) treba vyčítať vyplnené texty editačných polí
c) vyčítané texty uložíme do premenných
d) dáta z premenných zašleme pomocou SQL príkazu na SQL server
e) novo vytvorené dáta zobrazíme v mriežke v aplikácii
f) celá aplikácia sa pripraví na príjem nových dát

Teraz sa pokúsme na to pozrieť z pohľadu SQL:

a) na vytvorenie SQL príkazu použijeme komponentu *Query*
b) na dopĺňanie záznamov sa používa SQL príkaz *INSERT*
c) musíme použiť parametre na prenos dát z editačných polí do príkazu *INSERT*
d) vykonáme SQL príkaz
e) po uložení dát na SQL serveri na zobrazenie použijeme ďalší SQL príkaz *SELECT*

Keď sme už rozobrali návrhy z týchto pohľadov, vytvoríme telo procedúry TForm1.Button1Click.

Dnes z metodického hľadiska uvediem presný zdrojový kód tejto procedúry, ktorý si postupne rozoberieme.

Výpis celej procedúry je:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Query1.Close;
  Query1.SQL.Clear;
  Query1.SQL.Add('insert into kniha(id, nazov, autor, vydavatel, cena) values("", :nazov, :autor,
    :vydavatel, :cena)');

  Query1.ParamByName('nazov').AsString := Edit1.Text;
  Query1.ParamByName('autor').AsString := Edit2.Text;
  Query1.ParamByName('vydavatel').AsString := Edit3.Text;
  Query1.ParamByName('cena').AsString := Edit4.Text;

  Query1.ExecSQL;

  Query1.SQL.Clear;
  Query1.SQL.Add('select * from kniha');
  Query1.Open;

  Query1.Last;

  Edit1.Text:= '';
  Edit2.Text:= '';
  Edit3.Text:= '';
  Edit4.Text:= '0.00';

end;
```

Rozbor procedúry
Celá procedúra vychádza z obidvoch vyššie spomenutých hľadísk. Zároveň rešpektuje pravidlá programovacieho jazyka produktu Borland Delphi. Všimnime si, že sa skladá z niekoľkých dielčích častí. Zámerne som ich oddelil medzerami, ktoré samozrejme nemajú vplyv na výsledný kód.

Poďme si ich rozobrať riadok po riadku:

Query1.Close - Táto metóda (*method*) nastaví vlastnosť (*property*) **Active** na ***False***. Tým uzavrie dátový zdroj. Pred každou manipuláciou s dátovým zdrojom je nutné ho uzavrieť.

Query1.SQL.Clear - Táto vlastnosť komponenty *Query* vyčistí zoznam SQL príkazov, zviazaných s touto komponentou.

Query1.SQL.Add('insert into kniha(id, nazov, autor, vydavatel, cena) values('''', :nazov, :autor, :vydavatel, :cena)') - Táto funkcia pridá do vyššie vyčisteného zoznamu reťazec, ktorý obsahuje SQL príkaz *INSERT INTO*.
My už tento príkaz dobre poznáme a vieme, na čo slúži. Všimnime si však malé rozdiely oproti tomu, čo sme sa naučili. Zatiaľ sme pracovali s príkazmy SQL tak, že sme v časti **values** zadávali skutočné - reálne hodnoty, ktoré sa potom uložili v tabuľke.
Ako príklad uvediem:

INSERT INTO kniha(id, nazov, autor, vydavatel, cena) values('''', 'Linux - internet server', 'Satrapa P.', Neokortex','450.00')

V našej aplikácii sa však skutočné hodnoty nebudú zadávať priamo v SQL príkaze, ale ich budeme vpisovať do editačných polí, z kadiaľ ich chceme preniesť do SQL príkazu. Je jasné, že budú zakaždým inakšie, preto ich nemôžeme napísať do SQL príkazu "natvrdo".

Preto sa v časti **values** nachádzajú tzv. parametre. Je to obdoba premenných. Tie označujeme dvojbodkou pred menom parametra. Obsah tohto parametra sa naplní v ďaľšej časti procedúry. A len pre zopakovanie - prázdne úvodzovky na prvej pozícii **values** znamenajú, aby sám SQL server doplnil hodnotu položky *id* podľa autoinkrementačných pravidiel.

Query1.ParamByName('nazov').AsString := Edit1.Text;
Query1.ParamByName('autor').AsString := Edit2.Text;
Query1.ParamByName('vydavatel').AsString := Edit3.Text;
Query1.ParamByName('cena').AsString := Edit4.Text;
Metóda ***ParamByName*** komponenty *Query* spôsobí nastavenie parametra s konkrétnym menom. Táto metóda je veľmi vhodná na napĺňanie parametrov v tzv. runtime režime, teda za behu programu, nie pri jeho preklade.
Môžeme jednoducho povedať:
Parameter ***nazov*** sa naplní textovou hodnotou, ktorá sa rovná textu v editačnom poli komponenty *Edit1*.
Identicky to funguje aj v ostatných riadkoch.
Čo to znamená?
Keď sme vyplnili editačné polia na formulári, tie sa uložili do parametrov ***nazov, autor, vydavatel, cena***.
Keďže sa mená týchto parametrov zhodujú s menami parametrov príkazu *INSERT INTO* v časti **values** (pozor - dvojbodka nie je súčasť mena parametra, len jeho označenie - je to špecifikum Delphi, v iných programovacích jazykoch to môže byť inak!), pri zavolaní tohoto SQL príkazu sa obsahy parametrov rovnajú.

Query1.ExecSQL - zavolanie tejto procedury spôsobí vykonanie SQL príkazu. Najprv sa parametre SQL príkazu naplnia hodnotami parametrov z metódy *ParamByName* a zároveň sa vykoná.
Teda v tomto našom príklade sa parametre časti **values** príkazu *INSERT INTO* naplnia textami komponent *Edit1* až *Edit4*.
Príkaz sa vykoná. Na SQL serveri sa uloží nový záznam s takými položkami, rovnajúcimi sa obsahom editačných polí formulára.

Query1.SQL.Clear - Znova vyčistíme zoznam SQL príkazov.

Query1.SQL.Add('select * from kniha') - Tentokrát naplníme zoznam príkazom *SELECT * FROM kniha*.

Query1.Open - táto procedúra je rovnaká ako *ExecSQL*, len pre SQL príkazy SQL, ktoré vracajú hodnotu je vhodnejšie použiť *Open*. Spôsobí spustenie vyššie zadaného SQL príkazu. V našom príklade SQL server vráti výpis tabuľky **KNIHA**, ktorý sa zobrazí v komponente *DBGrid1*.

Query1.Last - táto procedúra nastaví kurzor na posledný záznam tabuľky. V našom prípade sme to urobili preto, aby sme vždy po doplnení záznamu a následnom stlačení tlačítka **Ulož** uvideli v mriežke novopridaný záznam.

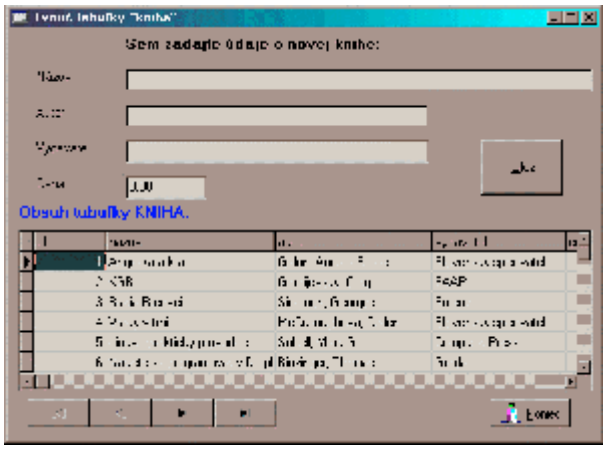
Edit1.Text:= '';
Edit2.Text:= '';
Edit3.Text:= '';
Tieto procedúry spôsobia vyprázdnenie editačných polí, aby sa pripravili na zadávanie nového záznamu.

Edit4.Text:= '0.00' - tento nastaví textovú hodnotu na tvar zápisu finančnej hodnoty.

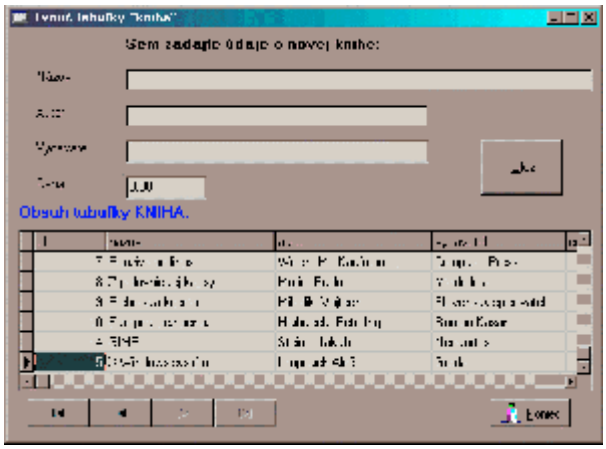
Takto sme sa dopracovali ku koncu procedúry ***TForm1.Button1Click***.

Ak sme vyššie uvedený zdrojový kód vpísali do súboru patriacemu k nášmu projektu (u mňa je to *unit1.pas*, ale vy si ho môžete samozrejme premenovať), znovu zkompilujeme náš projekt.

Spusťme projekt. Vyplňme správne meno a heslo loginového okna. Objaví sa formulár našej aplikácie tak, že kurzor v okne mriežky ukazuje na prvý záznam a všetky editačné polia sú prázdne, okrem *Ceny* (obr.č.18-6):

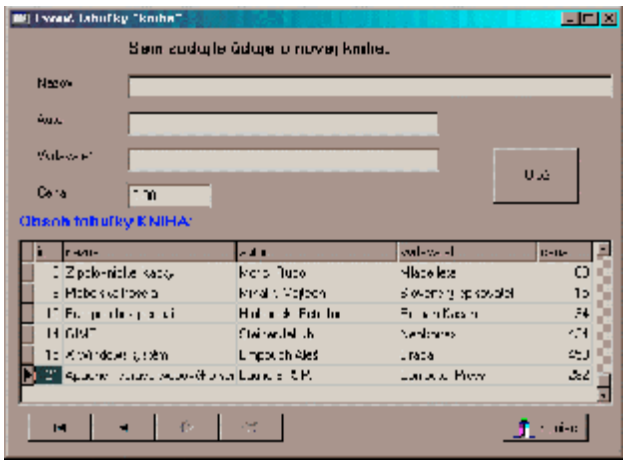


Prerolujme mriežkou a všimneme si, že sa údaje, ktoré sme zadávali na obr.č. 18-5 do tabuľky neuložili. To preto, lebo sme vtedy ešte nemali nadefinovanú činnosť tlačítka **Ulož**. Teraz presuňme kurzor na posledný záznam v tabuľke (obr.č.18-7):

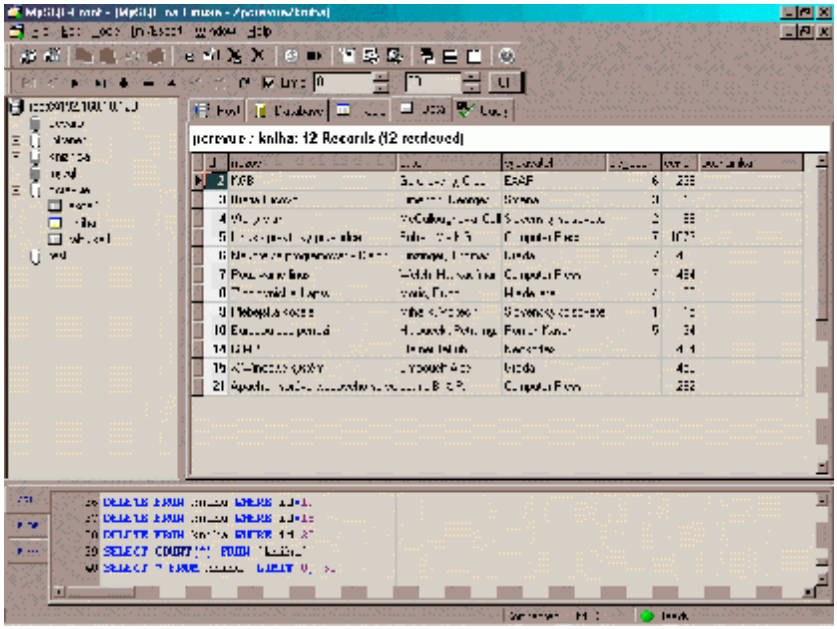


Vidíme, že má poradové číslo 15. (Ak sa dobre pozriete na obrázok, uvidíte, že čísla *id* nejdú postupne za sebou. Prečo? Spomeňme si, čo sme si hovorili o autoinkrementácii!)

Teraz znovu vyplníme editačné polia a kliknime na tlačítko **Ulož** (obr.č.18-8):



Čo sa stalo?
Editačné okná sa vyprázdnili, kurzor v mriežke sa posunul nižšie a objavil sa nový záznam, ktorého položky sú zhodné s tými, čo sme zadávali. Môžeme ukončiť činnosť aplikácie kliknutím na tlačítko **Koniec**.
Aby sme sa presvedčili, že nové dáta sú skutočne zapísané na strane SQL servera, môžeme použiť ľubovoľný prostriedok na spojenie so SQL serverom a pozrieť sa na obsah tabuľky **KNIHA** inak, teda nie cez našu aplikáciu napísanú v Delphi. Ja používam výborný ***MySQL-Front*** (obr.č.18-9):



(Všimnime si, že položky *cis_odd* a *poznamka* nie sú vyplnené. Je to tým, že sme nedefinovali ich vypĺňanie na ploche formulára).

Isto vás napadne, že táto aplikácia je už pomerne dokonalá, nedokáže však záznamy mazať, prípadne opravovať nejaké preklepy a pod. Taktiež by asi bolo výborné, aby sa číslo oddelenia nemuselo dopisovať, ale aby sa dalo vybrať z prednastavených možností.
Úpravy aplikácie na tieto potreby by boli veľmi podobné ako sme robili dnes. Iba by sa zmenili SQL príkazy. Ak chcete, môžete si to urobiť za domácu úlohu. Ak to zvládnete, a ja verím, že áno, dostanete plnohodnotnú aplikáciu a ste u cieľa.

Nie všetkým z nás vyhovuje spojenie pomocou ODBC. Nie na každom počítači je nainštalované alebo chodí bez problémov. Preto je v takýchto prípadoch vhodnejšie opustiť ODBC a ísť priamou cestou s využitím API. To dosiahneme pomocou iných komponent v Delphi.
O tom sa budeme rozprávať nabudúce. Zdrojové kódy dnešného príkladu, ako aj vyššie spomenutý program z obr.č.18-9 si môžete stiahnúť z mojej web stránky www.mior.host.sk.

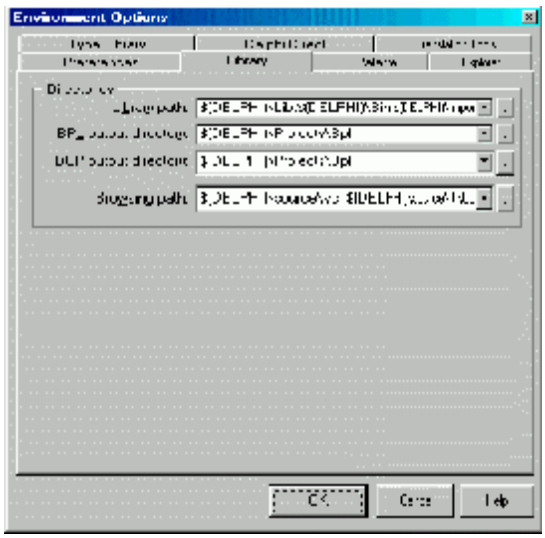
Malé veľké databázy II / 9.časť

Aj dnes sa do tretice povenujeme delfistom. Ostatní databázisti si však tiež môžu prečítať túto stať, možno v nej nájdu niekoľko inšpirujúcich podnetov. Veď filozofia programovania je rovnaká, iba príkazy a syntaktické zápisy sú v jednotlivých programovacích jazykoch odlišné.
Vráťme sa aspoň v myšlienkach k minulým častiam. Vytvorili sme aplikáciu v Borland Delphi, ktorá je schopná prezerať a dopĺňať vybratú tabuľku v príslušnej databáze. Na spojenie s konkrétnym MySQL serverom bolo použité ODBC, v našom prípade MyODBC.
Výhody a nevýhody spojenia cez ODBC alebo spojenia priamo, pomocou **API** - ***Application Programming Interface*** (aplikačné programovacie rozhranie) sme si vysvetlili nedávno. Dnes si ukážeme, ako takú aplikáciu, využívajúcu API, vytvoríme.

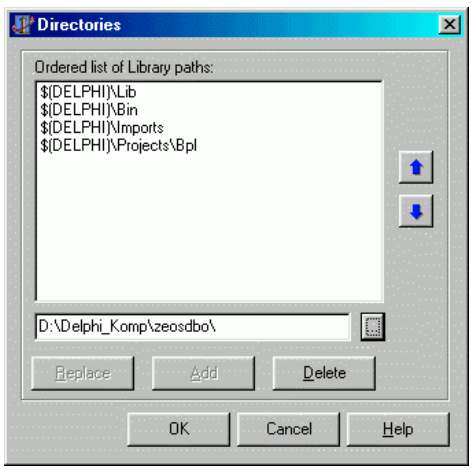
Čo potrebujeme
Ak sme v minulých častiach potrebovali MyODBC, tak dnes môžeme naňho zabudnúť. Naopak, budeme potrebovať API.
Ale ako také API vyzerá? API je pomyselné rozhranie, ktoré môže byť implementované rôznym spôsobom. Najčastejšie je to v tvare dynamickej knižnice, ktorá sa v systéme Windows označuje príponou ***.DLL*** (*dynamic loaded library = dynamicky nahrávaná knižnica*). „Oknoidní céčkari“ tieto knižnice veľmi dobre poznajú a veľmi často ich používajú. Aj delfisti majú svoje knižnice. Na „sieti sietí“ ich môžeme nájsť značné množstvo. Delphi je však produkt, postavený na využívaní komponent (vizuálnych či nevizuálnych) a preto aj my využijeme také API, ktoré je zakompilované v podobe komponent. Aj tých je samozrejme bezpočetné množstvo. Ja som si z určitého dôvodu (ktorý pochopíte neskôr) vybral komponenty od firmy **ZEOS**. Sú to komponenty, ktoré sú distribuované pod licenciou GNU a teda sú voľné a zdarma. Tieto príslušné komponenty si môžete stiahnúť z mojej web stránky www.mior.host.sk.

Inštalujeme ZEOS
Z internetu alebo vyššie uvedenej stránky si stiahneme balík komponent pre prácu s databázami s názvom *Zeos Database Objects* - **zeosdbo-5.3.0-beta5.zip**. Tento balík komponent je pre nás veľmi vhodný, lebo obsahuje komponenty pre Delphi 3, 4, 5, 6 a Kylix. Takže záleží už len na tom, ktorú verziu Delphi používame. Poďme na to metódou postupných krokov:

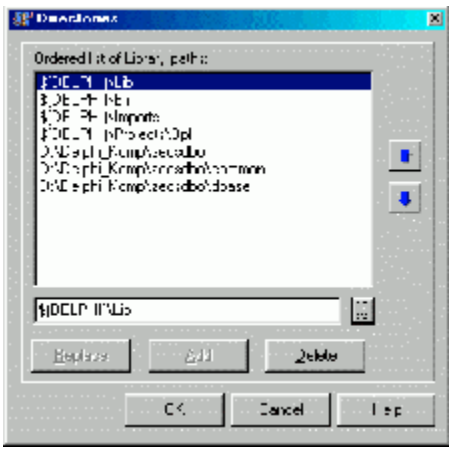
1) Do ľubovoľného adresára, kde zvyčajne inštalujeme komponenty, rozbalíme obsah zipovaného súboru **zeosdbo-5.3.0-beta5.zip**. Vznikne adreár **ZEOSDBO**. V ňom sa nachádzajú ostatné podadresáre a inštalačné súbory. Prečítame si príslušné súbory *readme* a *install*.
2) Súbor ***libmysql.dll***, ktorý sa nachádza v rozzipovanom adresári, prekopírujeme do adresára **Windows\System**.
3) Editujeme súbor ***zeos.inc***. Tu doporučujem zmeniť maximálne položku o definícii jazyka, teda **{$DEFINE CZECH}**. Ostatné zbytočne nemeníme!
4) V prostredí Delphi pridáme nastavenie cesty k príslušným súborom. Napr. v Delphi 5 to urobíme takto:
   a) spustíme Delphi
   b) Klikneme na položku menu *Tools - Enviroment Options* a v okne, ktoré sa objaví, otvoríme záložku *Library* - obr.č.19-1:



c) Klikneme na tlačítko s tromi bodkami vpravo od riadku ***Library Path:*** , čím otvoríme ďalšie okno *Directories* s položkami *Ordered List of Library paths:* (obr.č.19-2):



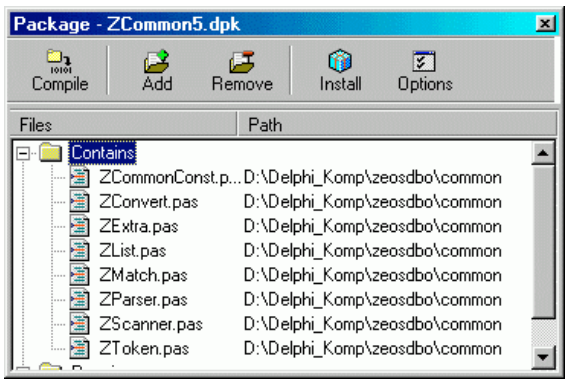
d) Kliknutím na tlačítko s tromi bodkami doplníme cesty k súborom komponent **ZEOSDBO** v tomto poradí:
XXX, **XXX\COMMON** a **XXX\DBASE**. **XXX** znamená úplnú cestu k adresáru ***ZEOSDBO***, teda napr.: ***D:\Delphi_Komp***\. Pomocou tlačítka **Add** takto doplníme všetky tri cesty. Výsledok vidíme na obr.č.19-3:



e) Stlačíme **OK** a vrátime sa do základnej obrazovky Delphi.

5) Zkompilujeme príslušné komponenty takto:

a) V menu Delphi klikneme na *File - Open*, nájdeme prvú sadu komponent s názvom ***ZCommonX.dpk***, kde ***X*** predstavuje číslo verzie komponent pre konkrétnu verziu Delphi, teda v mojom prípade ***ZCommon5.dpk***. Objaví sa okno balíčkovača *Package* (obr.č.19-4):



b) Klikneme na položku **Compile** a po úspešnej kompiácii na položku **Install**. Takto nainštalujeme aj ostatné komponenty presne v tomto poradí: ***ZDbwareX.dpk*** a ***ZMySqlX.dpk***.

c) Pozrieme sa, či sa nainštalovali všetky požadované komponenty na lištu komponent v Delphi - budú na konci lišty (obr.č.19-5):

Všimnime si, že sa naištalovalo týchto sedem komponent:

- **TZBatchSQL**
- **TZMonitor**
- **TZUpdateSQL**
- **TZMySQLDatabase**
- **TZMySQLTransact**
- **TZMySQLTable**
- **TZMySQLQuery**

Nezdajú sa nám názvy týchto komponent veľmi podobné, ako sú na lište ***DataAccess***?
Ale samozrejme, podobajú sa ako vajce vajcu, aj vlastnosti majú podobné a práve preto som si vybral túto sadu komponent **ZEOS**.

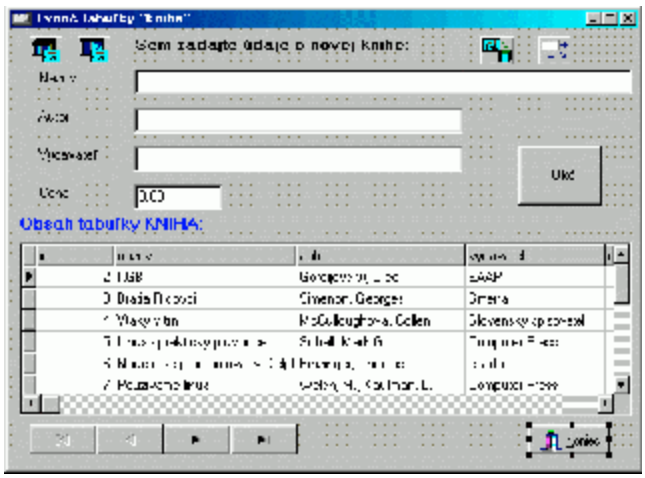
d) V adresári ***ZEOSDBO*** je dostupná dokumentácia, ktorú je veľmi vhodné si prečítať - v tvare HTML.

e) Môžeme ukončiť Delphi.

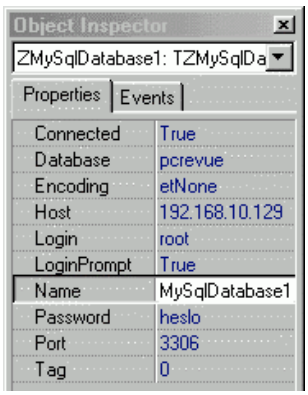
Vytvorenie aplikácie
Podstata aplikácie, ktorú budeme dnes vytvárať, je podobná tej z minulej lekcie. Použijeme iba iné komponenty z lišty **Zeos Access**.

1) Spustíme Delphi. Na formulár poukladáme jednotlivé komponenty z lišty **Zeos Access**:
   - **ZMySQLDatabase1**
   - **ZMySQLQuery1**
   - **ZMySQLTransact1**

a ostatné, nám už dobre známe komponenty z predchádzajúcej lekcie - **DataSource1**, **Edit1** až **Edit4**, **Label1** až **Label6**, **DBGrid1**, **DBNavigator1**, **Button1** a **BitBtn1** tak, ako je to na obr.č.19-6:



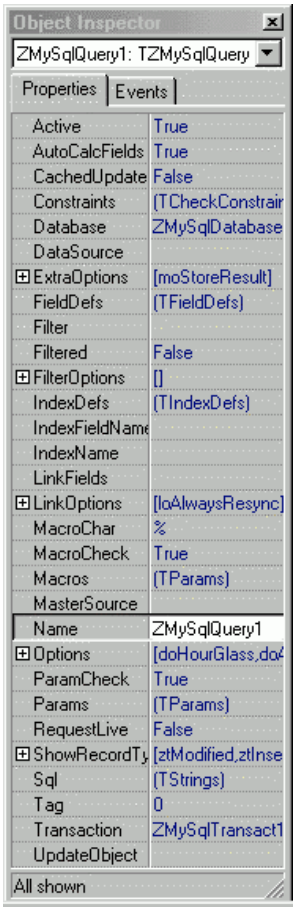
2) Nastavenie vlastností jednotlivých komponent
   Začneme komponentou **ZMySQLDatabase**, kde v *Object Inspectore* nastavíme vlastnosti tak, ako sú na obrázku č.19-7:



   Samozrejme, že ich nastavíme podľa svojho prostredia, teda *database* (názov databáze), *host* (meno alebo IP adresa počítača) ako aj *login* (meno užívateľa) či *password* (heslo) budú zodpovedať našim skutočnostiam.

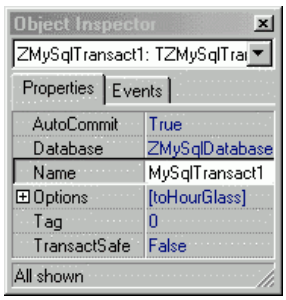
3) U komponety **ZMySQLQuery** sa zameriame na tieto vlastnosti:
   - ***Database***
   - ***Transaction***
   - ***Sql***

tak, ako je to na obrázku č.19-8:



Pozrime sa položku ***Sql***. Aká bude? No predsa dobre známa - ***select * from kniha***.

4) Komponentu **ZMySQLTransact** nastavíme veľmi jednoducho a presne podľa obrázku č.19-9:



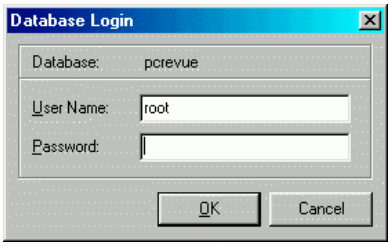
5) Ostatné komponenty sú rovnaké, ako v minulej časti.

6) Čo je v tejto aplikácii dôležité, je úprava procedúry, ktorá sa vykoná po kliknutí na tlačítko **Ulož**. Zdrojový text je v podstate veľmi podobný, ako v minulej časti, ale keďže sme použili namiesto komponenty **Query** komponentu **ZMySQLQuery**, jednoducho upravíme názvy príkazov z *Query* na *ZMySQLQuery* takto:

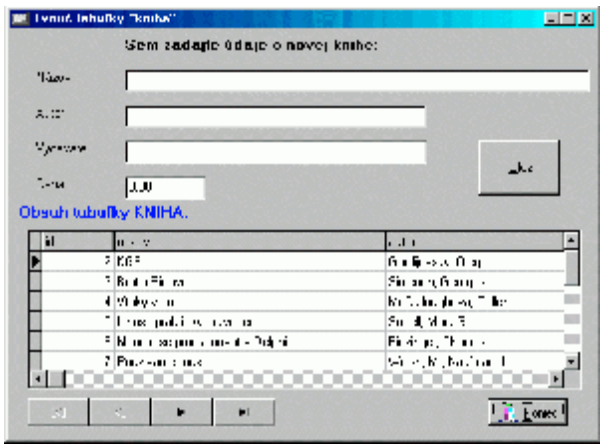
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ZmySqlQuery1.Close;
  ZmySqlQuery1.SQL.Clear;
```

```
ZmySqlQuery1.SQL.Add('insert into kniha(id, nazov, autor, vydavatel, cena) values("", :nazov,      :autor,
 :vydavatel, :cena)');


   ZmySqlQuery1.ParamByName('nazov').AsString := Edit1.Text;
   ZmySqlQuery1.ParamByName('autor').AsString := Edit2.Text;
   ZmySqlQuery1.ParamByName('vydavatel').AsString := Edit3.Text;
   ZmySqlQuery1.ParamByName('cena').AsString := Edit4.Text;


   ZmySqlQuery1.ExecSQL;

   ZmySqlQuery1.SQL.Clear;
   ZmySqlQuery1.SQL.Add('select * from kniha');

   ZmySqlQuery1.Open;

   ZmySqlQuery1.Last;
   Edit1.Text:= '';
   Edit2.Text:= '';
   Edit3.Text:= '';
   Edit4.Text:= '0.00';
end;
```

7) Zkompilujeme takto vytvorenú aplikáciu. Po kompilácii vznikne súbor s menom **PROJECT1.EXE** (jeho meno môžeme samozrejme zmeniť). Ak tento súbor spustíme, objaví sa úvodné logovacie okno (obr.č.19-10):



a po správnom vyplnení mena a hesla sa aplikácia pripojí k prednastavenému MySQL serveru (v mojom prípade k počítaču s IP adresou *192.168.10.129*), do stanovenej databáze (*pcrevue*) k danej tabuľke (*KNIHA*). Obsah tejto tabuľky sa zobrazí v mriežke - obr.č.19-11:



A môžeme pracovať.

Čo sme dosiahli
Vytvorili sme aplikáciu, ktorá nie je závislá na MyODBC driveri. Znalí delfisti by dokázali túto aplikáciu upraviť tak, aby vlastnosti komponenty **ZMySQLDatabase**, ako sú *Database*, *Host*, *Login* a *Password* boli konfigurovateľné priamo v aplikácii, teda aby neboli trvalo zakompilované ako teraz, ale aby ich bolo možné nastavovať podľa aktuálnej situácie. Tak by sme dostali naozaj univerzálnu aplikáciu, nezávislú na prostredí, adrese MySQL servera a podobne. Toto však už nie je cieľom tohto seriálu. Zdrojové texty aplikácie je možné stiahnúť z mojej web stránky www.mior.host.sk.

Týmto sme ukončili strednú školu databáz. Naučili sme sa nielen ovládať samotný MySQL server, vytvárať tabuľky, pridávať, mazať alebo upravovať záznamy z príkazového riadku, ale sme sa naučili pristupovať k dátam, uloženým na serveri „z diaľky“, teda cez niektorú inú aplikáciu. A to sme si mohli dokonca vybrať - či použijeme možnosti ODBC alebo API. A naučili sme sa niečo z teórie databáz, ako sa navrhujú databázy, čo je to normalizácia a podobne.

Čo nas čaká
Ako dobrí a usilovní študenti nechceme zakrnieť. Chceme sa naďalej rozvíjať, zlepšovať, naučiť niečo nové, aby sme boli medzi databázistami určitou špičkou. Preto postúpime na vysokú školu. A čo sa učí na takej vysokej škole? Najprv sa rozvinú znalosti zo strednej školy do hĺbky (možno by bolo správne v súlade s názvom školy povedať do výšky) a potom ešte niečo naviac. A tak sa pozrieme bližšie na indexy, ktoré majú svoju jedinečnú krásu, povenujeme sa mnohoužívateľským aplikáciam - teda, keď k jednému zdroju dát pristupuje naraz a súčasne niekoľko užívateľov. S tým je spojené zamykanie záznamov a podobné záležitosti. A aby sme sa nebáli o citlivé dáta, naučíme sa niečo z oblasti tranzakcií, replikácií a archivácií databáz.

A to najdôležitejšie. Prvý semester našej vysokej školy začína už nabudúce.

Okrem iného
Ako to vyznieva z vaších emailov, mnohým z vás učarovala možnosť používať MySQL na platforme Linuxu. Ale keďže ste sa (viacerí) doteraz venovali iba databázam a hlavne prostrediu Windows, neviete ako vstúpiť do sveta Linuxu. Chýba vám akýsi návod, čo s tým a ako. Preto vás pozývam na **Stretnutie s Linuxom**, ktoré začne už od budúceho čísla na inom mieste PCRevue. (Nahradí seriál o Sambe).