Malé veľké databázy - 1.časť

„Najväčšia cena je cena informácie“ povedal dávnejšie akýsi obchodník, a isto tým myslel tú správnu informáciu v správny čas. Tu správnu v nekonečnej zmeti všelijakých informácii, balastu aj skutočných perál. Kedysi, keď jediným komunikačným prostriedkom na väčšie vzdialenosti bol konský povoz královskej pošty, bolo informácii poriedko, a tak nebolo veľkým problémom udržať ich v kompetentnej hlave a správne vyhodnotiť. Avšak dnes, v dobe keď sa na nás valia informácie zo všetkých strán rýchlosťou svetla, je veľmi dôležité rôzne informácie niekde ukladať a vyhodnocovať na základe konkrétnych kritérií. Takýmto „skladom“ informácii hovoríme databázy.

Čo môže byť databázou?
Sú všade okolo nás, v zamestnaní, na ulici alebo v tej mliekárni na rohu ulice. Lebo účtovné programy, cestovný poriadok autobusov alebo zoznam tovaru v obchode nie je ničím iným jako skladom informácií, teda databázou.

A preto sa poďme pozrieť databázam bližšie na zúbky. Vysvetlíme si základné pojmy, povieme si niečo z histórie databáz a programov na prístup k nim. A kedže dnes vládnu svetu databázy typu SQL (čo to je, to sa dozvieme neskôr), budeme sa venovať jednej z nich, a to MySQL. Že ste o nej už počuli, ale neviete nič podrobnejšie? Tak práve vám je určený tento seriál. Túto databázu si nainštalujeme a čo je najzaujimavejšie, budeme s ňou pracovať na obidvoch najrozšírenejších operačných systémoch posledných dní – na Linuxe i na Windowsoch. A tak si prídu na svoje priaznivci oboch systémov. Okrem inštalácie sa spravovanie SQL serveru na obidvoch systémoch skoro nelíši, a tak nie je problémom „konvertovať“ k inej platforme. A naviac, nie je problém pristupovať k serveru na Linuxe z Windows alebo naopak, ba dokonca si taký programček v Delphi, prípadne v inom vyššom programovacom jazyku napíšeme. Že neviete programovať? Tak si údaje z databáze pripojíme do Excelu a tam si z nich urobime graf! A nakoniec si ukážeme, jako pomocou SQL serveru môžeme prezentovať svoje dáta na intranete alebo dokonca internete tak, že naši klienti budú potrebovať iba internetový prehliadač. Tak čo, ideme na to?

Trocha teórie.
Hned úvodom si povedzme, že nebudeme brať v úvahu rozdiel medzi dátami a informáciami. Budeme predpokladať, že všetky dáta sú pre nás zároveň informáciami, pretože ich dokážeme interpretovať a priradiť im zmysel.

Dáta sa ukladajú do tabuliek (tab.č.1). Naša cvičná tabuľka sa nazýva **KNIHA.**

ID	NAZOV	AUTOR	VYDAVATEL	TEMA	CENA	POZNAMKA
1	Angelika a kráľ	Golon Anne a Serge	Slovenský spisovateľ	román	56	
2	KGB	GordijevskyOleg	EAAP	faktografia	239	
3	Bratia Ricovci	Simenon Georges	Smena	krimi	18	
4	Vtáky v tŕni	MccIloughováCollen	Slovenský spisovateľ	román	66	
5	Linux-praktický pruvodce	Sobell Mark G.	Computer Press	odborná	1073	
6	Naučte se programovat v Delphi	Binzinger Thomas	Grada	odborná	439	
7	Používame Linux	Welsh M.a Kaufman L	Computer Press	odborná	494	

Databázová tabuľka je veľmi podobná tabuľke v textovom editore alebo v tabuľkovom kalkulátore, napr. *Excel, Quattro* a pod., alebo tej, čo si narysujeme doma na papier, aby ste zaznamenali určité informácie. Skladá sa z jednotlivých riadkov a stĺpcov. Riadkom hovoríme ***záznamy*** (vety) a stĺpcom hovoríme ***položky*** (polia). Výrazy v zátvorkách sú tiež správne, sú to ekvivalenty, s ktorými sa tiež môžeme v databázach stretnúť. My bude používať tie prvé výrazy.

Stĺpce tabuľky
Každý stĺpec (položka) má názov, ktorému sa hovorí atribút. Atribútom tretej položky je ‘AUTOR’, teda v tomto stĺpci sa budú nachádzať mená autorov jednotlivých kníh. Každá tabuľka musí obsahovať aspoň jeden stĺpec.

Riadky tabuľky
Zatiaľ čo položky majú rozdielne atribúty, záznamy obsahujú rovnaké položky. To znamená, že jednotlivé riadky majú rovnaké stĺpce. V našej tabuľke vidíme iba 7 záznamov, ale v tabuľke býva obvykle veľa riadkov. Môžeme sa stretnúť s tabuľkami, ktoré obsahujú až milióny riadkov, ale na druhej strane sú tabuľky, v ktorých nie je žiadny riadok. Takýmto tabuľkám hovoríme, že sú prázdne. Záznamy môžeme do tabuľky pridávať, alebo ich mazať, ale iba celé riadky. Taktiež výsledkom našich dotazov (lingvisti sorry, ale akosi mi slovo „dopyt“ nejde z úst!) bude jeden alebo skupina celých riadkov.

Hodnoty = dáta =informácie
V priesečníku riadku a stĺpca je konkrétna hodnota (aj prázdna hodnota je hodnota!). Táto hodnota reprezentuje určitý vzťah medzi záznamom a položkou. V priesečníku riadku popisujúceho knihu „Briatia Ricovci“ a stĺpca „CENA“ nájdeme sumu 18, čo predstavuje osemnásť vtedajších korún československých. (Skutočne, voľakedy stála kniha smiešne peniaze).

Súbor tabuliek, logicky vytvárajúcich určitý celok, tvorí ***databázu***. Tak ako každá tabuľka, tak aj databáza má konkrétne meno, ktorým sa prezentuje. Našu databázu pomenujeme **KNIZNICA**. V našej databáze máme zatiaľ iba jednu tabuľku **KNIHA**, ale môže ich tu byť viac, ktoré s danou tématikou projektu riadenia výpožičky kníh budú úzko súvisieť. Isto nás napadne, že by tu mala byť akási tabuľka o členoch - čitateľoch knižnice, napr. **CLENOVIA**, kde bu budú ich osobné údaje a aké knižky si požičali. Nepredbiehajme, k tomuto sa ešte dostaneme.

Databázový systém
Okrem rozčlenenia dát do jednotlivých tabuliek v rámci databáze máme k dispozícii ešte ***nástroje*** na manipuláciu s týmito dátami. Môžeme dáta vkladať, upravovať, meniť alebo vykonávať nad nimi rôzne operácie, ako napr. sčitovať, porovnávať, triediť do podskupín a podobne.
Zlúčením dát a nástrojov, pomocou ktorých vykonávame spomínané operácie, dostaneme ***databázový systém***.
Každý databázový systém musí obsahovať tieto základné nástroje pre:

* vytvorenie, vyhľadávanie, aktualizáciu a rušenie užívateľských dát
* definíciu štruktúry dát
* definíciu a zaistenie integrity dát
* zaistenie fyzickej a logickej nezávislosti dát

a prípadne nástroje pre:

* podporu práce viacerých užívateľov (definovanie prístupových práv)
* zálohovanie dát
* replikáciu dát

Fyzická nezávislosť dát znamená oddelenie zpôsobu fyzického uloženia dát (napr. na disku) od práce s nimi. Ak chceme pracovať s tabuľkou kníh, odkážeme sa na ňu pomocou jej názvu **KNIHA** a nemysíme sa zaoberať tým, kde je uložená na disku počítača a akým spôsobom sú fyzicky zaznamenané data v nej uložené.
O logickej nezávislosti dát hovoríme vtedy, keď zmena logickej štruktúry dát (napr. rozšírenie o ďalšie tabuľky alebo stĺpce v existujúcej tabuľke) nevyžaduje úpravu už existujúcich programov alebo dotazov pracujúcich s dátami.

Dátove typy
Už z našej jednoduchej tabuľky vidíme, že dáta obsiahnuté v rôznych stĺpcoch môžu byť rôzneho druhu. Je tu hodnota v stĺpci **CENA**, ktorá je zadávaná ako číslo a môžeme teda s ňou vykonávať číselné operácie, napr. sčítať, odčítať, násobiť a pod. Ďalej sú tu hodnoty v stĺpcoch **NAZOV, AUTOR, VYDAVATEL,** ktoré sú textového charakteru a môžeme ich porovnávať, hľadať podobné atď. Existujú však aj iné druhy hodnôt, ktoré sa môžu v tabuľke vyskytovať. Stĺpec **ID** je tak trochu zvláštny typ. Je síce číselný, ale s autoinkrementačnou vlasnosťou, teda databázový systém sám za nás pridá číslo o jedno vyššie, keď vložíme nový záznam do tabuľky. Odborne sa týmto druhom hodnôt hovorí ***dátové typy***. Základné dátové typy databázového systému MySQL, s ktorým budeme pracovať, sú v tab.č.2. Samozrejme, MySQL ich má podstatne viac. Keď budeme niektorý typ používať, vysvetlíme si ho.

Dat. typ	Popis
int	celé číslo v rozsahu -2 147 483 648 do 2 147 483 647
smallint	celé číslo v rozsahu -32 768 do 32 767
tinyint	celé číslo v rozsahu od 0 do 255
float	číslo s pohyblivou rádovou čiarkou
char(n)	textový reťazec dĺžky n (max. 255 znakov)
varchar(n)	textový reťazec max. dĺžky n (max. 255 znakov)
decimal(p)	desatinné číslo s p platnými číslicami
decimal (p,d)	desatinné číslo s p platnými číslicami a s d desatinnými miestami
money	peňažná čiastka
datetime	údaj o čase a dátume vo formáte RRRR-MM-DD HH:MM:SS
time	údaj o čase v tvare HH:MM:SS
date	údaj o dátume v tvare RRRR-MM-DD
blob	špeciálny typ pre ukladanie dlhých binárnych dát

V jednom stĺpci dátovej tabuľky sa môžu vyskytovať iba hodnoty rovnakého datového typu. Potom je zrejmé, že nemá zmysel požadovať vloženie textovej hodnoty do stĺpca **ID** alebo **CENA**. Preto hovoríme o dátovom type celého stĺpca tabuľky.

Nabudúce si povieme niečo väzbách v tabuľkách, o kľúčoch a indexoch. A nainštalujeme si datbázový systém.

Malé veľké databázy - 2.časť

V minulej časti sme sa venovali iba jednej tabuľke. Povedali sme si však, že v databáze môže byť (a obvykle býva) viac tabuliek. Hlavný význam databáz spočíva práve vo väčšom počte vzájomne previazaných tabuliek. Predstavme si teraz, že naša knižnica má viac oddelení, kde sú knihy uložené podľa jednotlivých žánrov. Prehľad týchto oddelení je v tabuľke č.3, ktorú si nazveme **ZANER**:

CIS_ODD	TEMATIKA
1	poézia
2	román
3	krimi
4	detská lit.
5	cestopis
6	lit. faktu
7	odborná lit.

Vráťme sa k našej prvej tabuľke s názvom **KNIHA**. Vidíme, že stĺpec *TEMA* sa dá nahradiť stĺpcom *CIS_ODD* z tabuľky **ZANER**. Potom bude naša opravená tabuľka **KNIHA** vyzerať takto:

ID	NAZOV	AUTOR	VYDAVATEL	CIS_ODD	CENA	POZNAMKA
1	Angelika a kráľ	Golon, Anne a Serge	Slovenský spisovateľ	2	56	
2	KGB	Gordijevsky,Oleg	EAAP	6	239	
3	Bratia Ricovci	Simenon, Georges	Smena	3	18	
4	Vtáky v tŕni	Mccloughová,Collen	Slovenský spisovateľ	2	66	
5	Linux-praktický pruvodce	Sobell, Mark G.	Computer Press	7	1073	
6	Naučte se programovat v Delphi	Binzinger, Thomas	Grada	7	439	
7	Používame Linux	Welsh,M.,Kaufman,L	Computer Press	7	494	

Nebudeme sa teraz zaoberať spôsobom, ktorým by sme upravili tabuľku v batabázi a prejdime rovno k výslednej podobe tabuľky **KNIHA**. K čomu je dobré, že sme vymenili textový zápis žánru za číslo riadku z inej tabuľky? Ak budeme pridávať ďalší záznam o novej knihe v našej knižnici, už nemusíme pracne vypisovať textový reťazec, napr. *odborná*, ale stačí zapísať príslušný číselný kód z tabuľky **ZANER**. Pri zapisovaní veľkého počtu kníh to podstatne zjednoduší a zrýchli napĺňanie tabuľky **KNIHA**.
Predstavme si, že by sa jedného dňa oddelenie *román* premenovalo na *próza*, čím by zahŕňalo aj novely, poviedky a iné. (Priznávam, že toto nie je z literárno - vedeckého pohľadu úplne čisté, ale ide o ilustračný príklad). Potom stačí opraviť iba jedno miesto v jedinej tabuľke **ZANER**. Keby sme si nechali pôvodnú tabuľku, museli by sme u každého záznamu, ktorý v stĺpci *TEMA* obsahuje slovo *román*, prepísať zameranie tematiky na *próza*. V našom prípade by sa jednalo iba o dva riadky z tabuľky **KNIHA**, ale čo by to bolo za knižnicu, keby sme mali iba dve knihy z tohto žánru, veď v reálnej situácii by to mohli byť stovky a stovky záznamov.
Takýmto istým spôsobom by sme mohli nahradiť stĺpec *VYDAVATEL*. Zistili sme, že do pomocných tabuliek vkladáme tie informácie, ktoré sa nie veľmi často menia a sú pre mnohé záznamy v hlavnej tabuľke spoločné. Potom sa v hlavnej tabuľke stačí odvolať na príslušné položky pomocnej tabuľky.
Vidíme, že takto vznikol medzi dvomi rôznymi tabuľkami určitý vzťah, a pre popis tohto vzťahu sa používajú tzv. ***identifikátory***.

Identifikátory riadkov
Úlohou identifikátoru je jednoznačne identifikovať jeden riadok v tabuľke. Identifikátor musí teda byť v rámci jednej tabuľky jedinečný - nesmú existovať dva riadky s rovnakou hodnotou identifikátoru. Teraz vieme, že stĺpec *ID* tabuľky **KNIHA** , ako aj stĺpec *CIS_ODD* tabuľky **ZANER** sú identifikátormi, lebo spĺňajú podmienku jedinečnosti.
Identifikátorom býva spravidla číselná hodnota, u ktorej máme istotu, že sa nikdy nezmení. U osôb to býva ich rodné číslo, číslo pasu alebo číslo občianskeho preukazu.

Dôvodom pre použitie čísiel miesto textových názvov býva úspora miesta a následne i zvýšenie rýchlosti spracovania. Textový reťazec totiž môže byť až 255 znakov (a teda aj bajtov) veľký, zatiaľ čo číslo môže mať veľkosť jedného bajtu. A pri vyhľadávaní je neskonale rýchlejšie porovnať jeden bajt ako 255 bajtov. Ak si domyslíme tabuľku o milióne záznamov, potom také vyhľadávanie v nevhodne vytvorenej tabuľke môže trvať aj veľmi dlhý čas. Zapamätajme si, že v niektorých prípadoch nestačí k jednoznačnej identifikácii riadku iba jedna hodnota. Aby sme mohli presne identifikovať daný riadok, musíme poznať hodnôt viac. Vtedy hovoríme o zložených identifikátoroch.

Primárny kľúč

Identifikátor je akýmsi kľúčom k danému riadku tabuľky, a preto mu hovoríme aj ***kľúč***. Ak má identifikátor okrem vlastnosti jedinečnosti aj minimálnu dĺžku, hovoríme mu ***primárny kľúč***, alebo aj primary key. Môžeme povedať, že stĺpec *ID* a *CIS_ODD* sú primárne kľúče vo svojich tabuľkách.

My vieme, že sa v tabuľke **KNIHA** odkazujeme do tabuľky **ZANER** pomocou stĺpca *CIS_ODD*. Ak sa teda primárnym kľúčom cudzej tabuľky vytvára odkaz v tabuľke **KNIHA**, je tento stĺpec tzv. *cudzím kľúčom* (foreign key).

Indexy

Okrem vlastných dát, ako je názov, autor, vydavateľ, cena, žáner knihy a podobne, sú v databáze uložené aj informácie, ktoré sú iba pomocné. Tieto informácie slúžia hlavne pre zrýchlenie spracovania našich požiadaviek. Medzi najviac používané doplnkové informácie k tabuľkám patria tzv. ***indexy***. S indexmi sa stretneme aj v našej knižnici.

Namiesto toho, aby sme museli prechádzať dlhé police a hľadať požadovanú knižku, stačí prehľadať malé kartičky v katalógu. Tieto kartičky, ktorým hovoríme aj indexy, sú spravidla uložené v šuplíku v registračnej skrini a sú zoradené podľa názvu knihy, čo nám veľmi zľahčuje vyhľadávanie. Samozrejme, že podľa názvu môžu byť zoradené aj knihy priamo v policiach na regáloch, ale to má niekoľko nevýhod. Manipulácia s ťažkými knihami je určite náročnejšia než s malými kartičkami. Ak by sme chceli pridať ďalšiu knihu doprostred police, museli by sme posunúť všetky nasledujúce knihy o jedno miesto ďalej. Na konci police by nemuselo byť dostatočné miesto a tak by sme museli prenášať poslednú knihu do ďalšej police. Naproti tomu vloženie malej kartičky do katalógu na správne miesto nie je vôbec náročné. A v prípade nedostatku miesta nie je problém presunúť niekoľko posledných kartičiek do inej priehradky.

Hlavnou výhodou použitia indexov však je, že ich môžeme vytvoriť aj niekoľko pre tie isté dáta. Môžeme mať jeden katalóg zoradený podľa názvu kníh, druhý podľa mien autorov a konečne tretí podľa žánru. Pri vyhľadávaní potom použijeme ten, ktorý sa najviac hodí pre naše potreby. Keď budeme vyhľadávať knihy o leteckom modelárstve (ach, kráásne časy), použijeme žánrový katalóg. Ak potrebujeme všetky diela od Karla Maya (ach, prekráásne časy), použijeme katalóg podľa autorov.

Aj keď hovoríme o knižnici a katalógoch s papierovými kartičkami, platia uvedené výhody a nevýhody v prípade použitia počítačov a elektronických médií. Aj v počítači musia byť dáta umiestnené na disku a ich fyzické radenie do určitého poradia a z toho vyplývajúce presúvanie zaberie určitý čas. Aj keď dáta vyhľadáva počítač, bude výsledok k dispozícii skôr (až o niekoľko rádov), ak bude „vedieť“, na ktorom mieste disku sú uložené záznamy o knihách s titulom začínajúcim na „M“. V opačnom prípade by musel postupne od začiatku prejsť riadok po riadku a kontrolovať názov titulu. V súčasnej dobe nie je potrebné vedieť presnú štruktúru indexov a spôsob práce s nimi. Stačí, ak vieme o ich existencii a o tom, že môžu výrazne urýchliť dobu potrebnú k získaniu požadovaných informácii z tabuľky.

Metadáta

Okrem vlastných dát a indexov sú v databáze uložené informácie popisujúce tieto dáta. V týchto „metadátach“ sú zaznamenané názvy existujúcich tabuliek, počty a názvy polí a ďalšie informácie nutné k tomu, aby sme mohli s databázou pracovať. Týmto metadátam sa niekedy hovorí slovník dát. Spravidla bývajú pre bežného užívateľa databáze neprístupné a preto sa nimi nebudeme ďalej zaoberať.

Administrátorské tabuľky práv

Okrem vlastných dát, indexov a metadát existujú v databázovom systéme špeciálne tabuľky, ktorým sa hovorí administrátorské tabuľky práv. V nich sú uložené informácie, kto a čo môže robiť s tou-ktorou databázou alebo tabuľkou. Nie všetci užívatelia databáze môžu robiť rovnaké operácie. Napr. hlavný knihovník môže dopĺňať záznamy do tabuľky **KNIHA**, zatiaľ čo bežný čitateľ môže túto tabuľku iba prezerať, aby zistil, aké knihy sú k dispozícii, nemôže však záznamy upravovať. A napr. účtovateľka knižnice nemôže dopĺňať záznamy, môže však meniť cenu jednotlivých kníh podľa ceny nákupu alebo precenenia.

Na toto slúžia práve tabuľky práv. Prístup do týchto tabuliek má správca databázového systému, ktorému sa hovorí aj administrátor. Môže to byť aj osoba, ktorej správca poskytne svoje práva alebo ich časť. V našom prípade budeme správcovia systému my.

Fakt, že dáta (všetky, či vlastné alebo administrátorské) verne zobrazujú reálny stav, ktorý popisujú, sa nazýva ***integrita*** alebo konzistencia dát. Základným predpokladom udržania dát v konzistentnom stave je kvalitne navrhnutá dátová základňa. Nekonzistencia medzi realitou a dátami, ktoré ju popisujú, môžu vznikať z týchto dôvodov:

- nedostatočná aktualizácia dát
  Dáta sa postupne stávajú neaktuálnymi. Stáva sa tak v prípade, že nedokážeme zaistiť pridanie nových záznamov do tabuľky, opravu hodnôt v existujúcich riadkoch a rušenie riadkov o už neexistujúcich objektoch reálneho sveta. Ako príklad poslúži strata určitej knihy. Ak neupravíme záznam o tejto knihe v tabuľke **KNIHA**, môže sa stať, že sa stále na ňu budú viazať objednávky k zápožičke a s tým súvisiace záznamy v iných tabuľkách. Pri štatistických údajoch o činnosti knižnice by sme mohli dostať nejednoznačné , ba dokonca nezmyselné výsledky. Preto je nutné pravidelne aktualizovať dáta. Aj keď sa jedná o triviálne úkony, môže udržiavanie dátovej základne v aktuálnom stave vyžadovať u rozsiahlych aplikácií nemalé úsilie, čas a peniaze

- referenčná integrita
  Tým, že máme údaje o knihách v jednej tabuľke, údaje o čitateľoch v tabuľke ďalšej a o vypožičaných knihách v tretej tabuľke, musíme dbať o to, aby sme pri rušení konkrétneho čitateľa vymazali nielen základné informácie o ňom, ale aj všetky jeho výpožičky. Obecne povedané, musíme vymazať všetky záznamy z ostatných tabuliek, ktoré sa na tohto čitateľa odkazovali. Ak by sme tak neučinili, neboli by sme neskôr schopní určiť, ku ktorému čitateľovi tieto údaje patria a dátová základňa by už nezobrazovala verne stav reality.

Relačný model dát

Keď sme hovorili o našich tabuľkách a vzťahoch medzi nimi, dali by sa popísať tieto vzťahy pomocou relačnej algebry. Týmto vzťahom potom hovoríme relácie, a z toho vyplýva, že používame ***relačný model dát***. Na tejto algebre je postavený jazyk SQL, ktorý je nosnou časťou našej práce. My sa tu nebudeme teraz zaoberať matematikou, ale toto uvádzam len pre úplnosť.

SRBD

Už vieme, čo sú to tabuľky a databázy. Prístup k údajom uložených v databáze obstaráva program, ktorému sa hovorí SRBD - Systém Riadenia Báze Dát. Tento krkolomný názov vznikol z anglického DBMS - DataBase Management System. Medzi SRBD patria také programy ako je Oracle, Informix, MS SQL Server, Progress. Nejedná sa o programy zrovna lacné, ich cena sa pohybuje v desiatkach či skôr v stovkách tisícok korún. Našťastie, aj na poli SRBD existujú programy šírené zdarma ako freeware - napr. PostgreSQL či MySQL, ktorý my budeme používať.

Jazyk SQL

V rokoch 1974 až 1975 prebiehal vo firme IBM výskum možností využitia relačných databáz. Pre tento projekt bolo nutné vytvoriť sadu príkazov, ktorými by sa relačná databáza ovládala. Tak vznikol jazyk ***SEQUEL*** (Structured English Query Language). Ako napovedá sám názov, bolo cieľom tvorcov vytvoriť jazyk, v ktorom by sa príkazy tvorili syntakticky čo najbližšie k bežnému (anglickému) jazyku. Tento jazyk sa veľmi rozšíril aj medzi ostatné firmy, zaoberajúce sa databázovými systémami a časom sa premenoval na ***SQL - Structured Query Language*** (štrukturovaný dopytovací jazyk) a štandardizoval sa. Preto väčšina príkazov, používaných napr. v MySQL, je funkčná aj v Informixe a iných systémoch. Hovorím väčšina, lebo, ako to už vo svete chodí, každý tvorca databáze vytvorí aj špecifickú skupinu príkazov len pre svoju databázu.

Architektúra klient - server

Databázové systémy bývajú rôznej architektúry. Najčastejšie sa používajú systémy s *centrálnym uložením dát*. Dáta sú uložené v databázach na konkrétnom centrálnom počítači a jednotlivé programy, pracujúce na lokálnych staniciach užívateľov v sieti, si zdielajú disk s uloženými dátami. To umožňuje, aby dáta boli centralizované a spravované na jednom mieste. Má to však určité obmedzenia, ktoré je nutné rešpektovať. Predstavme si, že máme databázu o veľkosti 20 megabajtov. Ak chce nejaký program pristúpiť k dátam v tejto databáze, potrebuje si k sebe po sieti natiahnuť do operačnej pamäte značnú časť z tejto databáze, pokiaľ nenájde príslušné informácie a nespracuje ich. Takto sa veľmi zaťaží prenosová schopnosť siete a ak budú naraz pristupovať viacerí užívatelia k tejto databáze, odozvy budú viac než katastrofálne. Preto sa systém zdieľania dát v tomto

spôsobe už veľmi nevyužíva. Z tohoto dôvodu sa vytvoril systém ***klient - server*** s jazykom SQL. Server pochádza z latinského servo = sluha, teda slúži ostatným klientom a vykonáva ich príkazy. Systém klient - server tiež využíva centrálne uloženie dát, avšak nad dátami operuje server SQL. Ten prijíma od jednotlivých klientov požiadavky (query), ktoré majú veľmi malú veľkosť niekoľkých bajtov, tieto spracuje a výsledok vracia po sieti klientovi. Ak je výsledkom napr. iba jeden riadok z tej 20 megabajtovej tabuľky, ktorý vyhovel požiadavke klienta, má odpoveď veľkosť iba pár bajtov. Takto sa informácie dostanú veľmi rýchlo k žiadateľovi, čo sa využíva hlavne v Internete a intranete. Ak takto pristupuje v lokálnej sieti naraz aj niekoľko užívateľov, sú odpovede aj tak veľmi rýchle. Moderné SQL systémy s architektúrou klient - server spracovávajú „on line“ požiadavky od tisícok užívateľov a títo ani nepostrehnú akési zdržanie vo svojej práci. Takýmto systémom sa hovorí, že sú to ***SQL servery***. Zapamätajme si, že SQL server aj klient nemusia, ale môžu bežať na tom istom počítači. Dokonca nemusia bežať na tej istej platforme. Server môže byť na Linuxe a klient na Windows, alebo naopak, ako si to ukážeme neskôr.
Ako som spomínal, my budeme pracovať s SQL serverom, ktorý sa nazýva ***MySQL***. Dnes je tento server (pre Linux) už free, čo znamená, že ho môžeme používať bez toho, aby sme platili licenčné poplatky. A samozrejme, taktiež ho môžeme slobodne použiť v našich komerčných aplikáciách. Tento server je dostupný jako pre platformu Linux, tak aj pre Windows. (Pre Windows môžete voľne použiť iba staršie verzie programu, za aktuálnu poslednú verziu pre Windows sa platia licenčné poplatky. Ale aj tie sú v porovnaní s inými komerčnými SQL servermi veľmi smiešne). Takže tí, ktorí majú možnosť pracovať s Linuxom, budú asi preferovať verziu pre Linux, ostatní, ktorým Linux ešte stále zaváňa exotikou, môžu pracovať na „oknoidnej“ verzii. Obe tieto verzie sú dostupné na rôznych cédéčkách, alebo na internetovej adrese www.mysql.com alebo na českom zrkadle www.mysql.cz. Na týchto adresách nájdete aj príslušného klienta pre požadovanú platformu. Posledná stabilná verzia nesie označenie 3.22.xx, verzie s označením 3.23.xx sú uvolnené ako alpha release, preto ich nebudeme používať. Na niektorých CD - diskoch s Linuxom, vydaných u nás alebo v Čechách, je verzia, ktorá je preložená aj s českým triedením. Toto však teraz nie je nutnosťou, ale pri tvorbe už ozajstného projektu sa to hodí. Pripravte si inštalačné súbory pre tú vašu platformu, nabudúce si ich nainštalujeme a začneme s nimi pracovať.

Malé veľké databázy/ 3.časť

Dnes si konečne MySQL server nainštalujeme. Verím, že ste niekde získali alebo stiahli z internetu príslušné súbory s SQL serverom a k nemu príslušným klientom. Zopakujme si, že server je tá časť, ktorá vykonáva naše príkazy a vracia nám požadované odpovede a klient je tá časť, cez ktorú tomu serveru tie príkazy zadávame a v ktorom požadované výsledky vidíme. Nainštalovať len samotný server je možné, lebo môžeme k nemu pristupovať aj inak, ako cez klienta. Napríklad môžeme komunikovať s SQL serverom aj cez webovský prehliadač (Netscape, Opera, Internet Explorer a iné) alebo z inej aplikácie, ako je Excel a podobne. Nainštalovať klienta bez servera zmysel nemá. Balíky MySQL (tak ako väčšina na Linuxe) sa označuje zvláštnym spôsobom, napr: *MySQL-3.22.27-1cs.i386*, kde číslica *3* znamená verziu použitých formátov súborov, číslo *22* je číslo jadra programu a číslo *27* je tzv. release verzia. *1cs* znamená úpravu na používanie čs. znakov a *i386* hovorí, že tento súbor je určený pre procesory Intel od 386-ky a vyššie. Posledná stabilná verzia je *3.22.xx*, kde na tom *xx* až tak veľmi nezáleží. Ak budete používať verziu *3.21.xx*, nemusia všetky príkazy fungovať korektne. Pripomínam, že verzie pre Windows môžete používať v zmysle shareware podmienok a ako free len tú, ktorá bola uvoľnená po uvedení novšej verzie (napr. *3.21.xx*). Pre Linux toto obmedzenie neplatí, ale zase sa nevypláca používať najposlednejšie verzie, (3.23.xx) ktoré bývajú spravidla ešte „nevychytané“!

Ako sme si povedali, MySQL server existuje pre platformu Windows aj pre Linux. Ukážeme si postupy inštalácie pre obidva systémy.

Inštalácia MySQL pre Windows

Kompletný balík, teda server aj klient, sa nachádza v jednom spakovanom súbore, napr. *mysql-shareware-3.22.34-win.zip*, čo je posledná stabilná shareware verzia pre 32-bitové Windows . Názov sa môže mierne líšiť podľa verzie programu. Jeho veľkosť je približne 5 MB.
Po rozzipovaní balíka spustíme program ***setup***, ktorý nás prevedie klasickou „oknoidnou“ inštaláciou, ktorú zvládne aj začiatočník. Môžeme si zvoliť cestu, kam sa MySQL nainštaluje. Pokým neinštalujeme MySQL do defaultného adresára *c:\mysql*, budeme potrebovať súbor *my.cnf*, ktorý je potrebné prekopírovať do koreňového adresára na disku *c*:. Tento získame premenovaním a upravením súboru *my-example.cnf*, ktorý nájdeme v adresári, kde sme inštalovali MySQL. Rovnako budeme súbor *my.cnf* potrebovať, ak z akéhokoľvek dôvodu našu inštaláciu neskôr presunieme do iného adresára. V oboch prípadoch je potrebné nájsť a upraviť podľa skutočnosti riadok s direktívou *basedir*. Na tomto riadku treba zrušiť komentár (zmazať znak #) a upraviť reálnu cestu k inštalácii MySQL. Ak sme ponechali inštaláciu na inštalačný program, a ten MySQL nainštaloval do adresára *c:\mysql*, súbor *my.cnf* nepotrebujeme a vyššie uvedené pokyny nevykonáme.
Po inštalácii je vhodné pridať do premennej *PATH* v súbore *autoexec.bat* adresár *bin* z inštalácie MySQL, aby sme mohli štartovať a zastavovať server a takisto aj spúšťať klienta a ostatné obslužné programy z príkazového riadku v ktoromkoľvek adresári.
Treba mať na pamäti, že sa inštalácia podľa verzie líši, napr. neumožňuje vybrať iný adresár ako *c:\mysql* a podobne (verzie 3.21.xx). Predpokladám, že ste dostatočne erudovaní a takéto malé odlišnosti vás nemôžu pomýliť. Na záver premenujeme program ***mysqld-shareware*** na ***mysqld***.
Ak sme úspešne vykonali inštaláciu, môžeme server poprvýkrát spustiť.

Spúšťanie a zastavovanie servera MySQL

Spôsob štartovania a zastavovania serveru sa líši podľa toho, či sme inštalovali na Windows 95/98 alebo na platformu NT.

Spúšťanie a zastavovanie vo Windows 95/98

Server spustime z adresára *mysql\bin* (alebo priamo, ak sme upravili premennú *PATH*) príkazom:
mysqld
alebo:
mysqld-shareware (niektoré verzie).
Server sa spustí na pozadí a je pripravený k činnosti.

Ak chceme server zastaviť, použijeme príkaz :
mysqladmin -u root shutdown

Spúšťanie a zastavovanie na Windows NT

Spúšťanie a zastavovanie na Windows NT môžeme vykonať dvojakým spôsobom - a to buď ako *konzolovú aplikáciu*, alebo ako *službu (service).* Ak server spúšťame ako konzolovú aplikáciu, použijeme príkaz:

mysqld –standalone
a zastavujeme už známym:
mysqladmin -u root shutdown.

Ak chceme používať MySQL na NT ako službu (service), pridáme server medzi služby príkazom:
mysqld –install
a štartujeme a zastavujeme server príkazmi:
net start mysql alebo ***net stop mysql***

A ešte jedna perlička. Ak spustíme server príkazom:
mysqld --language=slovak (v starších verziách je len voľba ***czech***)
bude s nami komunikovať v slovenskom jazyku. Odteraz dostaneme všetky chybové hlášky v zrozumiteľnejšej forme. Vyskúšajte!

Inštalácia MySQL na Linuxe

Pri inštalácii na Linuxe máme na výber z troch možností:

- užívatelia RedHatu a SuSE môžu použiť binárku vo formáte RPM
- inštalovať z vopred skompilovanej binárnej distribúcie
- použiť zdrojové texty a skompilovať si MySQL na vlastnej distribúcii Linuxu

Inštalácie binárnej distribúcie MySQL formátu RPM

Ak v našom počítači sídli Red Hat Linux, SuSE Linux alebo máme k dispozícii program rpm pre prácu s redhatovskými balíkmi, použijeme pravdepodobne tento typ inštalácie. Je najjednoduchšia a najrýchlejšia.
Čo budeme potrebovať? Tak napríklad balík servera *MySQL-3.22.27-1cs.i386.rpm* a balík klienta *MySQL-client-3.22.27-1cs.i386.rpm*.

Potom nás čaká jednoduchý krok. Pod X-Windows v našej obľúbenej aplikácii pre správu rpm balíkov (v GNOME je to *GNOrpm*, v KDE je to *kpackage*) tieto balíky nainštalujeme.
Druhou možnosťou je inštalácia z príkazového riadku. V adresári, kde máme uložené .rpm balíky spustíme príkaz:
rpm -Uhv MySQL-*.rpm
Takto sa naraz nainštaluje server aj klient, vytvoria príslušné databázy **test** a **mysql**. A naviac, server sa hneď spustí a beží na pozadí.
Zároveň sa nastavia štartovacie skripty tak, aby sa pri každom spustení Linuxu spustil aj MySQL server.

Inštalácia binárnej distribúcie MySQL formátu gz

V prípade, že náš Linux nevie pracovať s balíkmi rpm a my chceme inštalovať binárnu distribúciu, musíme si nájsť príslušný súbor. Ten má príponu *tar.gz* alebo *tgz*. Tieto prípony sú ekvivalentné a znamenajú, že tento súbor bol vytvorený balíčkovačom *tar* a pakovačom *gzip*.
Najprv sa rozhodneme, kam chceme MySQL inštalovať. Binárny súbor umiestnime o úroveň vyššie, teda ak chceme mať MySQL v adresári */usr/local/mysql*, umiestnime súbor do */usr/local*.
Uložený súbor rozbalíme pomocou príkazu:
tar zxvf meno súboru napr. ***tar zxvf mysql-3.22.27-pc-linux-gnu-i586.tar.gz***.
Týmto vznikne adresár *mysql-3.22.27-pc-linux-gnu-i586*, ktorý pre jednoduchší prístup premenujeme na *mysql*, alebo príkazom:
ln -s mysql-3.22.27-pc-linux-gnu-i586 mysql vytvoríme symbolickú linku *mysql* na daný adresár.

Vojdeme do adresára */mysql/scripts* a spustíme príkaz:
./mysql_install_db
ktorý vytvorí potrebné databázy **mysql** a **test**.
Tým je inštalácia hotová a server môžeme spustiť z adresára *mysql/bin* príkazom:
./safe_mysqld &

Znak „&“ (ampersand) zabezpečí, že sa server spustí na pozadí.

Inštalácia zo zdrojových textov

Ak sa rozhodneme skompilovať MySQL sami, musíme si zabezpečiť **zdrojové texty** programu. Upozorňujem, že kompilácia a inštalácia zo zdrojových textov je pomerne náročná pre Linuxového začiatočníka. Na druhej strane,

len pri tomto type inštalácie môžeme nastaviť parametre servera, ktoré by sme iným spôsobom nenastavili. Ako jeden z najdôležitejších parametrov považujem možnosť českého (teda skoro slovenského - až na niekoľko znakov) abecedného triedenia. To znamená, že takto skompilovaný server dokáže rozpoznať diakritické znaky a triediť podľa českej abecedy.
Súbor so zdrojovými textami si uložíme do ľubovoľného adresára a tento rozbalíme pomocou známeho príkazu:
tar zxvf mysql-3.22.27.tar.gz.
Po tomto kroku sa musíme rozhodnúť, kam chceme MySQL nainštalovať. Defaultným (štandardným = prednastaveným) adresárom je *usr/local*, toto však nemusí každému vyhovovať. Takisto sa musíme rozhodnúť, či chceme naše databázy umiestniť do defaultného adresára */usr/local/data* alebo ich umiestnime inde.
Teraz vstúpime do adresára, kde sme rozbalili zdrojové texty a spustíme príkaz:
./configure --prefix=[cesta_inštalovania_mysql] –localstatedir=[cesta_inštalovania_databáz] --with-charset=czech.
configure je konfiguračný príkaz, ***prefix*** a ***localstatedir*** nastavia cestu k serveru a databázam, ***with-charset*** nastaví triedenie v českom jazyku. Slovenský nie je k dispozícii.

Ak sa všetko vykoná ako má, teda bez chybových hlásení, je naša inštalácia pripravená ku kompilácii, ktorú spustíme príkazom:
make
K samotnej inštalácii stačí zadať:
make install
a MySQL máme v nami zadanom adresári. Odporúčam pridať do premennej *PATH* cestu ku spustitelnym súborom, teda *[cesta_inštalovania_mysql]/bin*.
Samotná kompilácia je náročná nielen na nervy kompilujúceho, ale hlavne na čas. Na takom dnes už priemernom PC s procesorom Pentium 233 MMX so 64 MB RAM trvá asi 30 minút, na slabších aj hodinu. Na silnejších počítačoch je to len niekoľko minút.

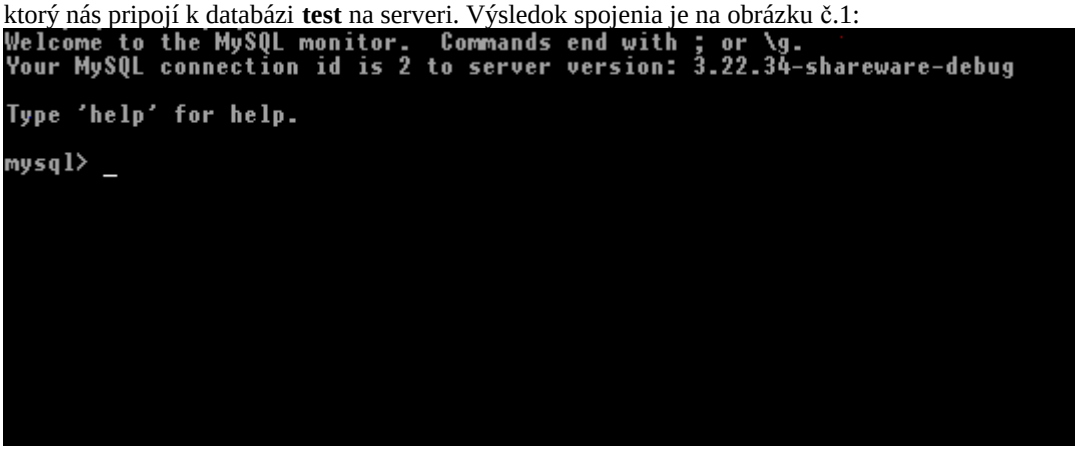
Po samotnej kompilácii je potrebné ešte vytvoriť databázy **test** a **mysql**. O toto sa postará skript ***mysql_install_db***, ktorý sa nachádza v rozbalenej distribúcii zdrojových textov a v adresári *scripts*.
Po dokončení skriptu máme k dispozícii MySQL server s užívateľom **root** bez hesla a riadkového klienta *mysqld*.

MySQL server spustíme pomocou druhého skriptu ***safe_mysqld &*** .

Prvé spojenie

Ak sme úspešne nainštalovali server aj klienta v hociktorej platforme ľubovoľným vyššie spomínaným spôsobom, môžeme spustiť server a vyskúšať prvé spojenie s ním.

Či v príkazovom riadku DOS-u alebo v textovej konzole Linuxu, odteraz budú príkazy pre prácu s MySQL serverom alebo riadkovým klientom rovnaké.
Zadáme príkaz:
mysql -u root test
ktorý nás pripojí k databázi **test** na serveri. Výsledok spojenia je na obrázku č.1:



Ak vidíme na monitore podobný obrázok, máme dôkaz, že SQL server beží. Klient *mysql*, nazývaný aj *MySQL monitor* sa zobrazí tzv. *promptom* **mysql>**, čo značí, že očakáva zadanie príkazu. Zoznam najzákladnejších príkazov získame po zadaní príkazu ***help***.

Aby bolo jasné, že zadávame príkazy v prostredí klienta *mysql*, budeme ich uvádzať týmto promptom mysql>, aby sme ich odlíšili od príkazov, zadávaných v operačnom systéme.
Ak chceme prácu s týmto riadkovým klientom ukončiť, zadáme príkaz
mysql> ***exit***
alebo
mysql> ***quit***.

Root

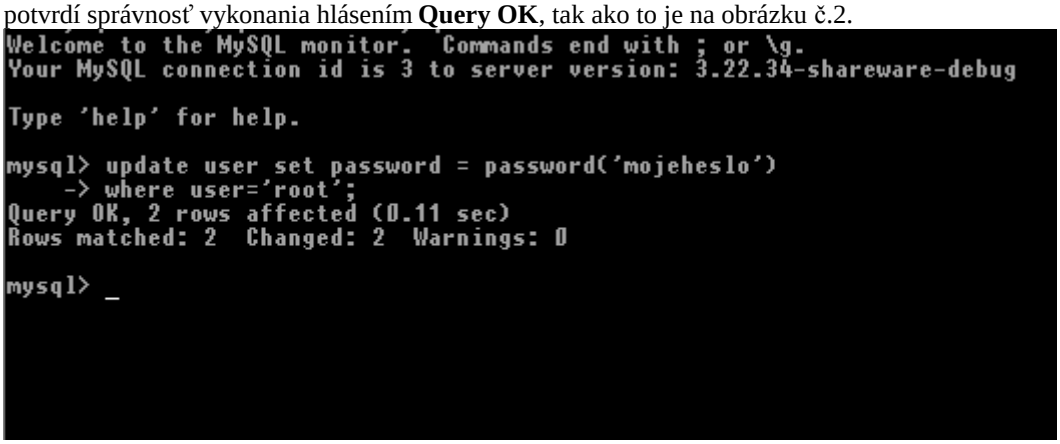
Ten, kto má v MySQL najvyššie postavenie a teda má plné práva k spravovaniu servera, sa nazýva **root**. Pozor, aj keď je jeho meno totožné s najvyšším správcom **root** na Linuxe, nejedná sa o tú istú funkciu a nemusí (ale môže) to byť ani tá istá osoba. Ak je na Linuxe administrátor root Janko Konôpka s heslom *Marienka*, môže byť v MySQL administrátorom root Ferko Malý s heslom *5bvD45*. Ale samozrejme nič nebráni tomu, aby Janko Konôpka bol zároveň rootom v MySQL, a heslo do Linuxu mal rovnaké ako do SQL servera alebo úplne iné.

Nastavenie hesla pre roota

Základným prvkom bezpečnosti je používanie hesiel pre užívateľov. MySQL sa štandardne inštaluje s vytvoreným užívateľom **root** bez hesla. Preto je potrebné heslo nastaviť. To dosiahneme pripojením k databáze **mysql** ako *root* príkazom:
mysql -u root mysql
Spustí sa klient *MySQL monitor* a v prompte zadáme príkaz:
mysql> ***update user set password=password('heslo_pre_roota') where user='root';***

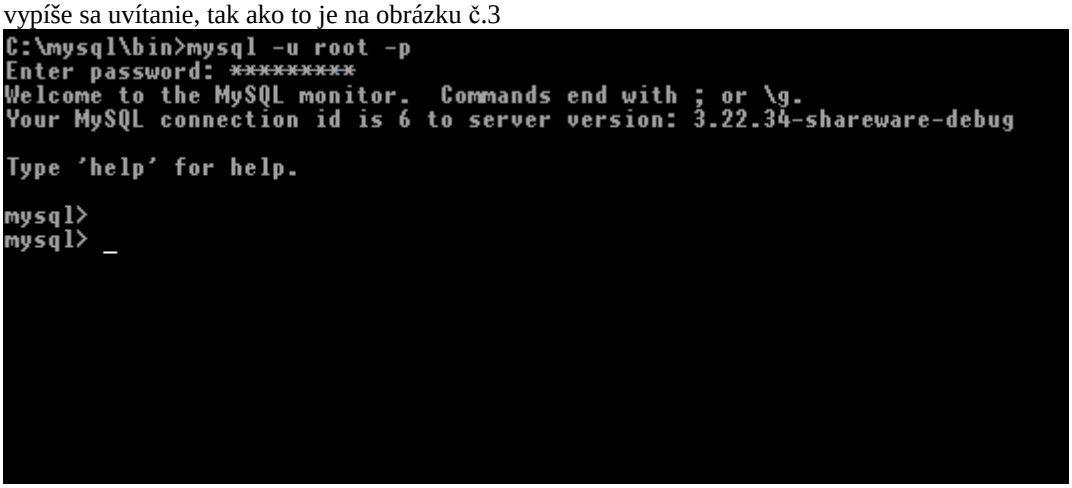
Za *heslo_pre_roota* si dosadíme naše heslo, ktoré chceme v spojení s rootom používať. V zásade by heslo malo byť také, aby ho nikto nebol schopný uhádnuť. Nie je vhodné používať ako heslo mená svojich blízkych, rôznych vecí a podobne. Najťažšie odhaliteľné heslá sú kombinácie náhodných znakov s číslicami, napríklad *fr45HC8a*. Upozorňujem, že MySQL tak ako Linux alebo Windows NT je *case-sensitive*, teda rozlišuje malé a veľké písmená. Teda heslo '*priklad_hesla*' nie je totožné s heslom '*PrIkLaD_HeSlA*'. Heslá, tak ako všetky textové reťazce, sa zadávajú v **apostrofoch** (znak **'**,nie v dvojitých úvodzovkách).

Pozor na znak **;** (bodkočiarka) na konci príkazu! Tento nie je súčasťou SQL príkazu, ale ho vyžaduje riadkový klient *mysql* a znamená niečo také ako „teraz vykonaj tento príkaz". Všetky príkazy, nielen SQL, zadávané v tomto klientovi sa musia ukončovať znakom **;**. Ak je príkaz, ktorý zadávame veľmi dlhý, stačíme klávesu **Enter** a klient vytvorí nový riadok označený šípkou **->**. Tu môžeme pokračovať a keď zapíšeme celý príkaz, nezabudnime na bodkočiarku. Stlačíme **Enter** a nami zadaný príkaz sa vykoná. Ak sme neurobili chybu, príkaz potvrdí správnosť vykonania hlásením **Query OK**, tak ako to je na obrázku č.2.



Význam jednotlivých inštrukcií vo vyššie uvedenom príkaze si objasníme neskôr.
Teraz je potrebné ukončiť klienta príkazom ***exit*** (alebo ***quit***) a v príkazovom riadku operačného systému spustiť príkaz:
mysqladmin reload
ktorý spôsobí načítanie nových prístupových práv. Ak sa chceme teraz pripojiť k serveru k databáze **test**, musíme použiť prepínač ***-p***:
mysql -u root -p test
alebo
mysql -u root -pheslo_pre_roota (spolu s písmenom „p"- bez medzery!).
Ak sme zadali prvý tvar príkazu, systém sa spýta na heslo hláškou **Enter password:**, ktoré musíme zadať z klávesnice. Pri zadávaní sa z dôvodu bezpečnosti heslo vypisuje hviezdičkami, to preto, keby sa niekto pozeral

cez rameno. Preto musíme byť pozorní pri ťukaní do klávesnice. Mám jednu dobrú radu: nepoužívajme heslo, v ktorom sú znaky „y“ a“z“. To preto, že nevieme, či je nastavená klávesnica QWERTY alebo QWERTZ. Ak zadáme nesprávne heslo, pokus o spojenie zlyhá a server vydá chybovú hlášku. Ak sme zadali heslo správne, vypíše sa uvítanie, tak ako to je na obrázku č.3



Základné programy pre prácu s MySQL

Ak sa pozrieme do adresára, kde sme inštalovali MySQL, zistíme, že sa tu nachádzajú ešte ďaľšie programy. Teraz sa stručne oboznámime s tými, ktoré budeme najčastejšie používať:

- mysqld - samotný SQL démon - server
- mysql - riadkový klient MySQL monitor
- mysqladmin - administrátorská utilita. Vytvára a maže databázy, načítava tabuľky práv, vypisuje verziu, procesy a status servera.
- mysqldump - exportuje (dumpuje) databázu do súboru
- mysqlimport - nahráva dáta z textového súboru do príslušných tabuliek
- mysqlshow - zobrazuje informácie o databázach, tabuľkách, stĺpcoch a indexoch
- mysqlmanager - grafický klient (len vo Windows XX)

Základné databázy

Ako sme si povedali, po úspešnej inštalácii sa vytvoria v systéme MySQL dve základné databázy **test** a **mysql**. Databáza **test** je klasická databáza, ktorú systém pripravil pre naše cvičné pokusy. Databáza **mysql** je databáza, v ktorej sú uložené administrátorské tabuľky prístupových práv. Ako sme skôr povedali, tu sa nastavujú užívatelia a ich práva pre prácu s rôznymi databázami a tabuľkami v nich. Zatiaľ čo do databáze **test** môže pristupovať každý užívateľ MySQL, s databázou **mysql** môže operovať iba root. Databáza **test** je zatiaľ prázdna, teda neobsahuje žiadne tabuľky. V databáze **mysql** sa nachádzajú tzv. „***grant tables*“**, ktorých počet sa líši v jednotlivých verziách programu. V starších verziách sú tri najzákladnejšie tabuľky - **user**, **db**, **host**. Podľa verzie sa okrem týchto tabuliek nachádzajú ešte tabuľky **tables_priv** a **columns_priv** alebo **func**. Ak chceme vedieť, aké tabuľky máme, v mysql klientovi zadáme skupinu príkazov:

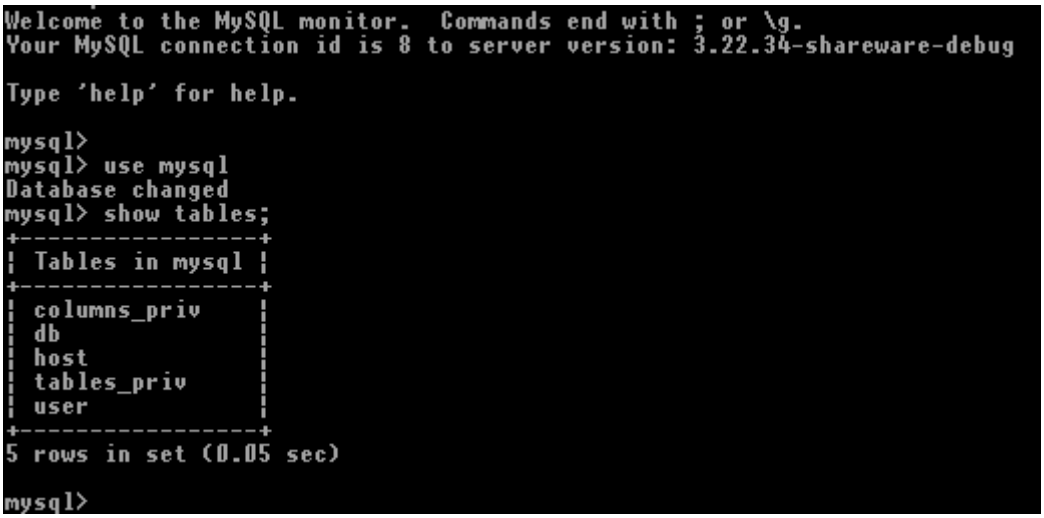
mysql> ***use mysql;***

ktorý povie systému, že budeme pracovať s databázou **mysql**.

Príkazom

mysql> ***show tables;***

vypíšeme všetky tabuľky v databáze **mysql**, tak ako to je na obrázku č.4.



Význam jednotlivých tabuliek si vysvetlíme v ďalšej časti seriálu.
Nabudúce si ukážeme, ako sa tvoria, napĺňajú a prezerajú naše známe tabuľky **KNIHA** a **ZANER**.

Poznámka:
Tí, ktorí majú prístup k Internetu, môžu nájsť ďalšie informácie na www stránke **www.cevaro.sk**. Nájdu tu linky na spomínané súbory, ako aj cvičné príklady, ktoré budeme postupne vytvárať.

Malé veľké databázy – 4.časť

Už vieme, ako nainštalovať, spustiť a zastaviť MySQL server, ba vieme sa aj prihlásiť jako root a zmeniť si heslo, takže nám už nič nebráni, aby sme s ním začali pracovať. Ako sme si povedali, na komunikáciu so serverom používane sadu SQL príkazov. Na zadávanie týchto príkazov môžeme použiť už známeho klienta MySQL Monitor (program *mysql*) alebo sadu pomocných programov. Ak sa chceme o možnostiach toho-ktorého pomocného programu dozvedieť viac, spustíme ho s parametrom *–help* alebo *–h*. V stručnej nápovede zistíme základné parametre daného programu. Pre názornosť budeme príkazy pre klienta **mysql** označovať promptom **mysql>**, bez tohoto promptu uvažujeme pomocné programy v prostredí príslušného operačného systému. Taktiež budeme príkazy písať malými písmenami, lebo klient *mysql* akceptuje aj malé písmená.

Základné príkazy SQL
MySQL obsahuje množinu príkazov podľa normy SQL92, my však budene zatiaľ využívať iba základnú skupinu, ktorá je popísaná v tabuľke č.4-1.

Príkaz	Popis
create table	vytvorenie tabuľky
drop table	zmazanie tabuľky
alter table	úprava štruktúry tabuľky
insert into	vloženie záznamu do tabuľky
update	úprava záznamu v tabuľke
delete	vymazanie záznamu v tabuľke
select	výpis z tabuľky

Z tabuľky je zrejmé, že na základnú činnosť so serverom postačuje sedem príkazov. Azda najdôležitejším príkazom je **SELECT**. Jednotlivé príkazy a ich možnosti si objasníme postupne, ako budeme vytvárať našu databázu.

Naša prvá databáza
Ako sme si povedali, budeme pracovať s databázou **KNIZNICA** a s tabuľkami v nej uloženými. Vieme, že MySQL server po inštalácii obsahuje dve základné databázy - **MYSQL** a **TEST.** Preto musíme databázu **KNIZNICA** vytvoriť sami.
Na vytvorenie databáze použijeme program **mysqladmin** s voľbou **create *meno_databáze***. Takže zadáme:

mysqladmin create *kniznica*

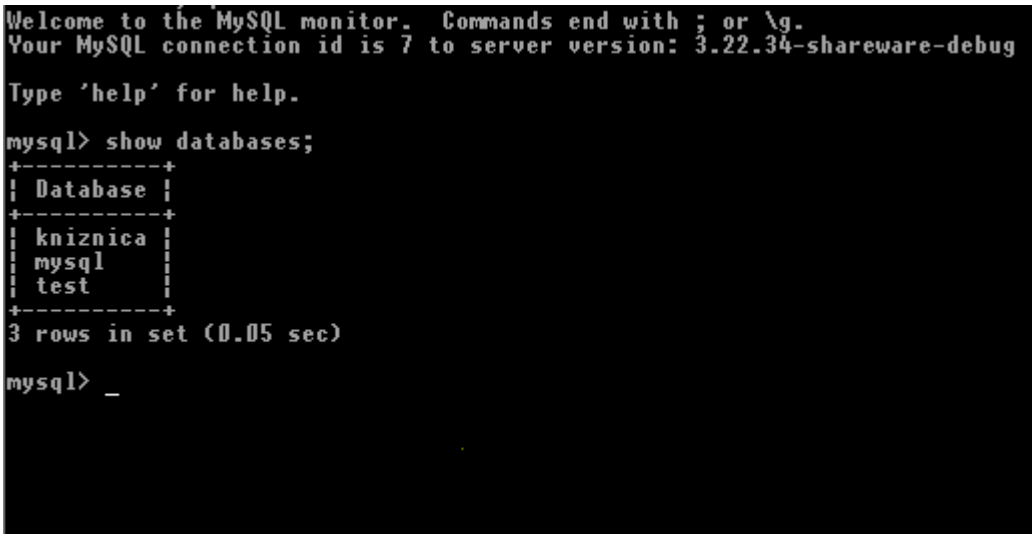
Aby sme sa presvedčili, či naozaj všetko prebehlo v poriadku, použijeme daľší pomocný program

mysqlshow

kde na obrazovke uvidíme výpis jednotlivých databáz, vytvorených na našom serveri. Môžeme použiť aj klienta *mysql*, v ktorom zadáme príkaz pre klienta:

mysql> show databases;

Odpoveď servera na tento príkaz by mala byť taká, ako na obr.č. 4-1:



Tabuľky
Teraz máme vytvorenú databázu **KNIZNICA**. Zatiaľ je prázdna, ale vieme, že bude obsahovať (dočasne iba dve) tabuľky **ZANER** a **KNIHA**. Tabuľka sa vytvorí príkazom:

create table *meno_tabulky (položka1 typ_položky1, položka2 typ_položky2,...,položkan typ_položkyn)*

kde *meno_tabulky* je konkrétne meno, napr. ZANER a v zátvorkách sa uvádza meno položky a jej daný typ.

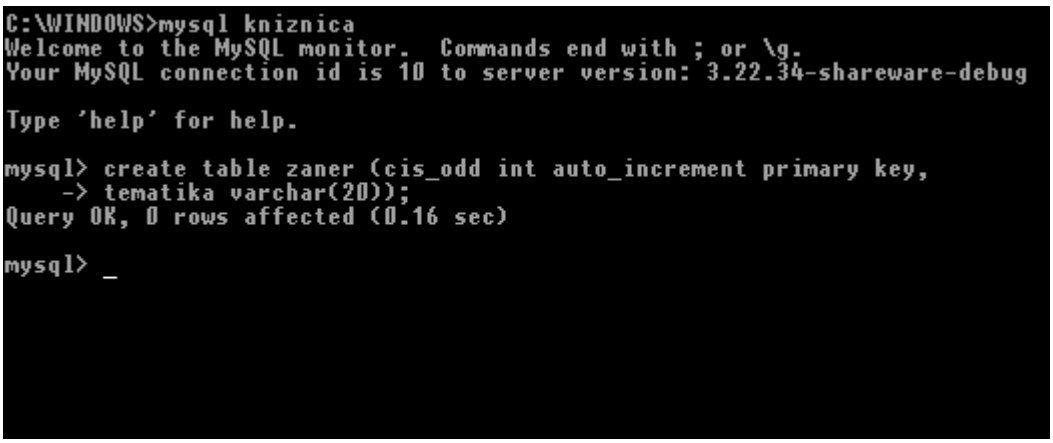
Tabuľka **ZANER** má iba dve položky – **cis_odd**, ktorá je typu int a **tematika**, ktorá je typu varchar o počte znakov do 20. Z konštruktívneho hľadiska je vhodné, aby prvá položka **cis_odd** bola zároveň primárnym kľúčom a aby mala autoinkrementačnú schopnosť. To znamená, aby sa vždy po pridaní nového záznamu automaticky zväčšila jej hodnota o 1.

Aby sme vytvorili tabuľku **ZANER**, spustíme klienta mysql s parametrom *kniznica*, čím sa automaticky nastavíme do prostredia databázy KNIZNICA. Odteraz príkazy, ktoré budeme zadávať, sa budú týkať databáze **KNIZNICA** a tabuliek v nej. Vytvorenie samotnej tabuľky vykonáme príkazom:

mysql > create table zaner (cis_odd int auto_increment primary key, tematika varchar(20));

Ak je riadok príkazu veľmi dlhý, stlačíme Enter a pokračujeme na novom riadku za šípkou. Spomeňme si, že príkaz klienta sa vykoná až po zadaní bodkočiarky a stlačení Enter!

Ak sme neurobili v zápise žiadnu chybu, server potvrdí vykonanie príkazu hláškou **Query OK**, tak ako je to na obrázku č.4-2:



Podobným spôsobom si vytvoríme aj tabuľku **KNIHA**. Polia tabuľky budú mať tieto názvy a typy, ako sú popísané v tabuľke Tab.č4-2:

Názov poľa	Typ poľa
id	int
nazov	varchar(40)
autor	varchar(30)
vydavatel	varchar(25)
cis_odd	int
cena	decimal(7,2)
poznamka	varchar(25)

Zadaním príkazu:

```
mysql > create table kniha (
        -> id int auto_increment primary key,
    -> nazov varchar(40),
    -> autor varchar(30),
    -> vydavatel varchar(25),
    -> cis_odd int,
        -> cena decimal(7,2),
         -> poznamka varchar(25));
```

sme vytvorili druhú tabuľku s názvom **KNIHA**.

Typ *id int auto_increment* a *varchar* už poznáme, typ *decimal(7,2)* znamená, že je to číselná hodnota na sedem miest, z čoho dve sú za desatinnou čiarkou (v počítačovom prostredí bodkou). Maximálna hodnota tohto výrazu je teda 99999.99, čo je myslím na cenu knihy postačujúce.

Aby sme sa presvedčili, že sme skutočne vytvorili tabuľku **KNIHA**, spustíme program:

```
mysqlshow kniznica
```

čím zobrazíme všetky tabuľky v databázi **KNIZNICA**.

Môžeme zadať príkaz pre klienta:

```
mysql > describe kniha;
```

Tak dostaneme štruktúru tabuľky tak, ako je na obr.č.4-3:

```
mysql> create table kniha (
    -> id int auto_increment primary key,
    -> nazov varchar(40),
    -> autor varchar(30),
    -> vydavatel varchar(25),
    -> cis_odd int,
    -> cena decimal(7,2),
    -> poznamka varchar(25));
Query OK, 0 rows affected (0.11 sec)

mysql> describe kniha;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      |      | PRI | 0       | auto_increment |
| nazov     | varchar(40)  | YES  |     | NULL    |                |
| autor     | varchar(30)  | YES  |     | NULL    |                |
| vydavatel | varchar(25)  | YES  |     | NULL    |                |
| cis_odd   | int(11)      | YES  |     | NULL    |                |
| cena      | decimal(7,2) | YES  |     | NULL    |                |
| poznamka  | varchar(25)  | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
7 rows in set (0.11 sec)

mysql>
```

Takýmto spôsobom si môžeme overiť, či sme tabuľku nadefinovali správne.

Ak by sme z akéhokoľvek dôvodu potrebovali tabuľku vymazať, použijeme príkaz:
mysql > drop table *meno_tabuľky;*
Po použití tohto príkazu je tabuľka nenávratne stratená, takže ho používajme s rozvahou!

Napĺňanie tabuliek
Ak sme úspešne vytvorili tabuľky, môžeme ich začať napĺňať vhodnými informáciami.
Na napĺňanie tabuliek sa používa príkaz **INSERT INTO** v takomto tvare:

insert into *meno_tabuľky (meno_1_položky, meno_2_položky, meno_n_položky)*
VALUES *(hodnota_1_položky, hodnota_2_položky, ..., hodnota_n_položky)*

Ak budeme dopĺňať položky v presnom poradí tak, ako sú nadefinovalé v tabuľke, nemusíme v zápise príkazu vypisovať názvy jednotlivých položiek pred slovom *VALUES* (=hodnoty). Tým si trochu zjednoduchšíme zápis.

Zadáme teda:
mysql> insert into zaner values (0, 'poezia');

Prečo sme na prvú pozíciu zadali nulu?
Keď sme tvorili tabuľku **ZANER**, určili sme, že stĺpec *CIS_ODD* bude mať autoinkrementačnú schopnosť. Preto nie je potrebné zadať do stĺpca *CIS_ODD* príslušné poradové číslo, napr. 1,2,3.... Ak zadáme v každom pridaní záznamu pomocou **insert into** nulu, server sám za nás priradí prvé voľné poradové číslo. Keď zadávame pridanie ďalších záznamov, znova na prvú pozíciu zapíšeme nulu, teda:

mysql> insert into zaner values (0, 'roman');

Pripomínam, že reťazce sa zadávajú do apostrofov alebo do úvodzoviek. Keďže niektoré operačné systémy neuznávajú v textoch úvodzovky, my budeme používať univerzálnejší apostrof.
(Poznámka: Možno nás napadne, či môžeme zadávať textové reťazce s diakritikou. Odpoveď je trochu komplikovaná. Ak prostredie operačného systému umožňuje písať a zobrazovať znaky s diakritikou na obrazovke, môžeme túto možnosť využiť. Ale ak nemáme MySQL server preložený z podporou triedenia podľa národnej abecedy, dostaneme v príkaze SELECT niekedy nežiadúcu odpoveď. Ako mi je známe, preklad samotného servera je možný iba v prostredí Linux, zatiaľ čo binárne súbory pre Windows nie sú skompilované s národnou podporou. Nie je to neriešiteľné, ale zatiaľ v týchto cvičných aplikáciách budeme používať texty bez diakritických znamienok.)

Ak sme nespravili žiadnu chybu, SQL server potvrdí vykonanie príkazu hláškou:
Query OK, 1 row affected (0.22 sec)
čo znamená asi toľko, že query - dopyt bol správny, bol ovplyvnený jeden riadok a koľko času trvalo vykonanie tohoto query. Takto doplníme aj ostatné záznamy do tabuľky **ZANER**:

mysql> insert into zaner values (0, 'krimi');
mysql> insert into zaner values (0, 'detska lit.');
mysql> insert into zaner values (0, 'cestopis');
mysql> insert into zaner values (0, 'lit. faktu');
mysql> insert into zaner values (0, 'odborna lit.');

Všimnime si, že sme nulu použili aj v ostatných zápisoch.

Vyhľadávanie záznamov
Ako sa presvedčíme, aké dáta máme uložené v tabuľke?
Na to použijeme príkaz **SELECT**.
Vieme, že úlohou príkazu SELECT je vybrať z danej tabuľky určité informácie na základe stanovených kritérií. Je to jeden z najmocnejších príkazov v SQL štruktúre. Jeho mnohým parametrom sa budeme postupne venovať, zatiaľ nám stačí vedieť najjednoduchší a zároveň najobecnejší zápis:

mysql> select * from *meno_tabuľky*;

Select znamená výber, * znamená všetko, *from* odkiaľ a za *meno_tabuľky* dosadíme už konkrétne meno. V našom prípade príkaz ***select * from zaner*** znamená : vyber všetko z tabuľky ZANER a výsledok bude vyzerať tak, ako na obrázku č.4-4:

```
mysql> select * from zaner;
+---------+-------------+
| cis_odd | tematika    |
+---------+-------------+
|       1 | poezia      |
|       2 | roman       |
|       3 | krimi       |
|       4 | detska lit. |
|       5 | cestopis    |
|       6 | lit. faktu  |
|       7 | odborna lit.|
+---------+-------------+
7 rows in set (0.06 sec)

mysql>
```

Mazanie záznamu
Na výmaz konkrétneho záznamu v určitej tabuľke sa používa príkaz:

delete from *meno_tabuľky* where *podmienka*

Podmienka, nasledujúca za slovom *WHERE* určuje, ktorý záznam bude zmazaný. Ak ani jeden záznam nesplňuje podmienku, žiadna zmena sa nevykoná.

Ak by sme teda chceli vymazať štvrtý riadok z tabuľky ZANER, príkaz by vyzeral asi takto:

mysql> delete from zaner where cis_odd = 4;

V prípade, že by sme rozhodovali podľa textového poľa TEMATIKA, podmienka by vyzerala takto:

mysql> delete from zaner where tematika = 'detska lit.';

Zmena záznamu
Na zmenu obsahu určitého záznamu sa používa príkaz **UPDATE**. Tento sme už raz použili, a to na zmenu hesla pre roota v tabuľke **USER** v databáze **MYSQL**, spomínate si?

Syntaktický zápis príkazu ***update*** je:

update *meno_tabuľky* set *položka = nová_hodnota* where *podmienka*

Ak by sme chceli upraviť šiesty záznam tabuľky ZANER na „faktografia“ namiesto „lit. faktu“, vykonali by sme to príkazom:

mysql> update zaner set tematika = ‘faktografia’ where cis_odd=6;

alebo

mysql> update zaner set tematika = ‘faktografia’ where tematika=’lit. faktu’;

Jeden aj druhý zápis je správny, záleží len na stanovení podmienky, určujúcej, ktorý záznam sa upraví.
Ak podmienke vyhovie viac záznamov, dôjde k zmene u všetkých záznamov, ktoré vyhoveli podmienke. Ak nevyhovie ani jeden záznam z tabuľky, ostane tabuľka bezo zmien.

Pre našu ďaľšiu činnosť si naplňte aj druhú tabuľku KNIHA aspoň niekoľkými záznamami, aby sme sa nabudúce mohli pohrať s mnohými variantami príkazu SELECT. Tu uvidíme silu SQL.
Aby sme si precvičili to, čo sme si dnes vysvetlili, ale zároveň nepoškodili nevhodným zásahom pracne vytvorené databázové základy budúceho projektu knižnice, vypracujeme si cvične domácu úlohu!

Domáca úloha:

1) Vytvorte si cvičnú databázu SKUSKA
2) V nej vytvorte tabuľku OSOBY, ktorá bude obsahovať tieto stĺpce a ich typy:
   por_cislo int auto_increment primary key
   meno varchar(30)
3) Naplňte tabuľku vhodnými dátami, (môžete vyskúšať aj zadávanie a zobrazovanie znakov s diakritikou
4) Vypíšte obsah tabuľky
5) Zmažte niekoľko záznamov podľa vami stanovenej podmienky
6) Vložte jeden nový záznam
   (Čo zaujímavé ste zistili???)
7) Zmeňte ľubovoľný záznam príkazom *update* a prezrite si výsledok
8) Čo sa stane, ak zadáte neúplný príkaz:
   delete from osoby
9) Zrušte tabuľku OSOBY
10) Zrušte databázu SKUSKA (*mysqladmin drop.....*)

Malé veľké databázy / 5.časť

V úvode si objasníme problémy, ktoré vznikli pri domácej úlohe z minulej časti. Pozrime sa bližšie na 6.príklad z úlohy: Čo sa stane, keď po zmazaní niektorého záznamu znova vložíme jeden nový záznam?
Odpoveď znie: Nový záznam sa nevloží na koniec ostatných záznamov, ale server nájde prvé voľné miesto po zmazanom zázname a využije ho. Ak už neexistuje voľné miesto po zmazaných záznamoch, pridá nový vkladaný záznam na koniec všetkých záznamov. Toto je veľmi dôležité uvedomiť si pri operáciách mazania a vkladania, inak by sme mohli dostávať nežiadúce výsledky.
8.príklad je ešte nebezpečnejší: Ak zadáme **: delete from *meno_tabuľky*** bez podmienky, zmaže sa **celá tabuľka!** A ako už vieme, príkaz *delete* je deštrukčný a prídeme o všetky dáta v príslušnej tabuľke.
Takže pozor na presné zadávanie príkazov!
Toľko k domácej úlohe.
Teraz si spusťme klient-monitor ***mysql*** s parametrom *kniznica*, aby sme mohli operovať s našimi tabuľkami.

Príkaz SELECT

SQL, a teda aj MySQL ponúka veľa spôsobov, ktoré slúžia na manipuláciu s dátami v databáze. Zvyšujú tak komfort obsluhy a kvalitu práce. Jedným z najznámejších, najužitočnejších a najpoužívanejších príkazov jazyka SQL je istotne príkaz ***SELECT*** (výber). Pomocou tohto príkazu môžeme vytiahnuť z databáze každý dôležitý údaj a vyriešiť tak veľa situácii v získaní tej správnej informácie.
V závere minulej časti sme si ukázali najzákladnejší tvar príkazu SELECT na prezeranie dát v našich tabuľkách KNIHA a ZANER v databázi KNIZNICA:

mysql> select * from zaner;

Výsledkom tohto príkazu je výpis č.5-1:

```
mysql> select * from zaner;
+---------+--------------+
| cis_odd | tematika     |
+---------+--------------+
|       1 | poezia       |
|       2 | roman        |
|       3 | krimi        |
|       4 | detska lit.  |
|       5 | cestopis     |
|       6 | lit. faktu   |
|       7 | odborna lit. |
+---------+--------------+
```

Príkaz SELECT toho však dokáže podstatne viac. Popis všetkých funkcií by obsiahol desiatky strán, preto sa zameriame iba na tie najpoužívanejšie. (Všetky možnosti príkazu SELECT sa nachádzajú v príslušnej dokumentácii k danému SQL serveru.)
Základný popis príkazu SELECT v MySQL je následovný:

SELECT {*zoznam výstupných položiek* } FROM {*zoznam tabuliek*}
//WHERE {*podmienky*}//
//GROUP BY{*zoznam položiek*}//
//HAVING {*skupinová podmienka*}//
//ORDER BY {*meno položky ako podmienka zotriedenia*}//

Zoznam výstupných položiek môže byť množina názvov jednotlivých stĺpcov tabuľky alebo aj tzv. ***agregačné funkcie***. Čo sú agregačné funkcie si ukážeme nižšie.
Zoznam tabuliek sú názvy tabuliek v danej databázi.
Parametre uvedené v lomítkach nie sú povinné.
Skúsme si aplikovať tieto podmienky prakticky na našej databáze **KNIZNICA**.

Vypísanie niektorých položiek

Ak chceme vypísať zoznam všetkých položiek (stĺpcov) danej tabuľky, použijeme znak ***** - hviezdičku. Hviezdička symbolizuje slovo „všetko“. Výsledok sme si ukázali vyššie.

Ak chceme vypísap len niektoré stĺpce z danej tabuľky, nahradíme hviezdičku vypísaním názvov požadovaných položiek v príkaze:

SELECT ***položka_1, položka_2, ...položka_n*** **FROM** ***meno_tabuľky***

Ak zadáme príkaz:

mysql>select id, nazov, autor from kniha;

dostaneme výsledok, kde sú zobrazené iba stĺpce **id, nazov a autor**, ostatné sú ignorované. Zobrazí sa teda zredukovaná tabuľka ako je na výpise č.5-2:

```
mysql> select id,nazov,autor from kniha;
+----+------------------------------+-----------------------+
| id | nazov                        | autor                 |
+----+------------------------------+-----------------------+
|  1 | Angelika a kral              | Golon, Anne a Serge   |
|  2 | KGB                          | Gordijevsky, Oleg     |
|  3 | Bratia Ricovci               | Simenon, Georges      |
|  4 | Vtaky v trni                 | McCulloughova, Collen |
|  5 | Linux - prakticky pruvodce   | Sobell, Mark G.       |
|  6 | Naucte se programovat v Delphi | Binzinger, Thomas   |
|  7 | Pouzivame linux              | Welsh, M., Kaufman, L. |
+----+------------------------------+-----------------------+
```

Agregačné funkcie

Agregačné funkcie sú typy dopytov, ktoré spracovávajú hodnoty z celých stĺpcov tabuľky. Zadávame ich do zoznamu výstupných položiek za príkaz SELECT.
V MySQL existujú tieto najpoužívanejšie agregačné funkcie:
SUM( ) - súčet numerických hodnôt v danom stĺpci
MIN( ) - nájdenie minimálnej hodnoty v danom stĺpci
MAX( ) - nájdenie maximálnej hodnoty v danom stĺpci
COUNT( ) - počet hodnôt (záznamov) v danom stĺpci
AVG( ) - aritmetický priemer numerických hodnôt v danom stĺpci

Vyskúšajme si tieto funkcie.

SUM( )

Ak by sme chceli vedieť, aká je hodnota (hodnota = súčet cien jednotlivých kníh) našej knižnice. Pre zjednodušenie predpokladajme, že máme iba po jednom kuse z každej knihy.
Použijeme funkciu ***SUM(cena)*** takto:

mysql> select sum(cena) from kniha;

a dostaneme tento výpis č.5-3:

```
mysql> select sum(cena) from kniha;
+-----------+
| sum(cena) |
+-----------+
|   2385.00 |
+-----------+
```

Že sa nám nepáči názov stĺpca *sum(cena)*? Nevadí, ak by naša účtovníčka nepoznala tieto pojmy, môžeme pre ňu upraviť príkaz *select* takto:

mysql> select sum(cena) as 'Hodnota knih' from kniha;

Slovko **as** spôsobí, že premenuje stanovenú funkciu (v tomto prípade „*sum(cena)*“) na stanovený text, napr. *Hodnota knih*. Potom dostaneme výpis č.5-4:

```
mysql> select sum(cena) as 'Hodnota knih' from kniha;
+--------------+
| Hodnota knih |
+--------------+
|      2385.00 |
+--------------+
```

No a to už je zrozumiteľnejší výpis.

MIN( ), MAX( )

Veľmi podobným spôsobom môžeme zistiť minimálnu alebo maximálnu cenu kníh v našej knižnici. Použijeme na to funkcie *MIN* a *MAX* s využitím formulky *AS 'konkrétny_text'*:

mysql> select min(cena) as 'Najmensia cena', max(cena) as 'Najvacsia cena knih' from kniha;

Výsledok je na výpise č.5-5:

```
mysql> select min(cena) as 'Najnizsia cena',
    -> max(cena) as 'Najvacsia cena knih'
    -> from kniha;
+----------------+--------------------+
| Najnizsia cena | Najvacsia cena knih |
+----------------+--------------------+
|          18.00 |             1073.00 |
+----------------+--------------------+
```

Ak sa pozrieme do našej tabuľky **KNIHA**, vidíme, že tieto vyselektované hodnoty sú skutočne správne.

AVG( )

Ak by nás zaujímala priemerná cena kníh v našej knižnici, požijeme funkciu *AVG( )* takto:

mysql> select avg(cena) as 'Priemerna cena knih' from kniha;

a dostaneme výpis č.5-6:

```
mysql> select avg(cena) as 'Priemerna cena knih'
    -> from kniha;
+--------------------+
| Priemerna cena knih |
+--------------------+
|          340.714286 |
+--------------------+
```

Count( )

Ak sa chceme dozvedieť, koľko že to máme teraz v knižnici evidovaných kníh, použijeme funkciu *COUNT( ),* ktorá zisťuje počet položiek v danom stĺpci.
Zadajme príkaz:

mysql> select count(*) as 'Pocet knih' from kniha;

a tu je výsledok (výpis č. 5-7):

```
mysql> select count(*) as 'Pocet knih'
    -> from kniha;
+------------+
| Pocet knih |
+------------+
|          7 |
+------------+
```

Poďme sa teraz pozrieť na „druhý koniec“ príkazu SELECT:

Formulácia podmienky WHERE

Formaláciu podmienky ***WHERE*** už trocha poznáme aj z iných príkazov SQL. Použili sme ju pri mazaní (*delete*) a upravovaní (*update*) jednotlivých záznamov v minulej časti seriálu. Aj pri príkaze SELECT spĺňa rovnakú úlohu - vyselektujú sa len tie záznamy, ktoré podmienke vyhovujú, ostatné záznamy sa ignorujú.
Obsahom podmienky môžu byť:

- operácie s textovým reťazcom (rovnosť, nerovnosť, podobnosť)
- matematické operácie (rovnosť, nerovnosť, väčší než, menší než)
- logické operácie (AND, OR, NOT)

Operácie z textovým reťazcom

Pri vyhľadávaní s formulkou *WHERE* s textovým reťazcom použijeme tento všeobecný zápis:

SELECT ***položka_1, položka_2, ...položka_n*** **FROM** ***meno_tabuľky***
WHERE ***názov_filtračného_stĺpca*** **OPERAND** ***hľadaný_reťazec***

Za ***názov_filtračného_stĺpca*** dosadíme názov stĺpca, podľa ktorého stanovujeme podmienku. Ak chceme vyhľadávať podľa názvu knihy, použijeme **NAZOV**, ak podľa ceny, použijeme **CENA**.
Za ***OPERAND*** dosadíme:
„ =“ (*rovná sa*), ak sa má porovnať presný text podmienky,
„<>“ (*nerovná sa*), ak sa nemá vôber rovnať hľadanému reťazcu
LIKE (*podobnosť*), ak sa má hľadaný reťazec podobať.

Za ***hľadaný_reťazec*** dosadíme stanovený text do apostrofov.
Tu je niekoľko príkladov:

= (rovná sa)

Ak by sme chceli vyhľadať názov, autora a cenu kníh, ktorých VYDAVATEL je vydavateľstvo Computer Press, použijeme príkaz:

mysql> select nazov, autor, cena from kniha where vydavatel = 'Computer Press';

Na výpise č.5-8 vidíme, že stanovenej podmienke vyhoveli dva záznamy:

```
mysql> select nazov, autor, cena from kniha
    -> where vydavatel = 'Computer Press';
+---------------------------+-----------------------+---------+
| nazov                     | autor                 | cena    |
+---------------------------+-----------------------+---------+
| Linux - prakticky pruvodce | Sobell, Mark G.       | 1073.00 |
| Pouzivame linux           | Welsh, M., Kaufman, L. |  494.00 |
+---------------------------+-----------------------+---------+
```

<> (nerovná sa)

Naopak, ak by sme chceli vypísať všetky záznamy o knihách, len nie tie, ktoré sú z vydavateľstva Computer Press, zapíšeme podmienku takto:

mysql> select nazov, autor, cena from kniha where vydavatel <> 'Computer Press';

Výsledkom bude výpis zvyšných piatich záznamov, tak ako na výpise č.5-9:

```
mysql> select nazov, autor, cena from kniha
    -> where vydavatel <> 'Computer Press';
+-------------------------------+-----------------------+--------+
| nazov                         | autor                 | cena   |
+-------------------------------+-----------------------+--------+
| Angelika a kral               | Golon, Anne a Serge   |  56.00 |
| KGB                           | Gordijevsky, Oleg     | 239.00 |
| Bratia Ricovci                | Simenon, Georges      |  18.00 |
| Vtaky v trni                  | McCulloughova, Collen |  66.00 |
| Naucte se programovat v Delphi | Binzinger, Thomas     | 439.00 |
+-------------------------------+-----------------------+--------+
```

LIKE

Ak nevieme presne názov knihy, nemôžeme použiť operand = . Ak vieme, aké slovo alebo časť slova daný názov obsahuje, požijeme ***LIKE*** a tzv. *divoké znaky*, ktorým hovorievame aj *žolíky* (jokes). Tieto v textových reťazcoch fungujú podobne ako žolíky v kartách - nahrídzajú iné znaky.
Divoké znaky sú v SQL databázach dva :
% - precento, ktoré nahrádza ľubovoľnú skupinu znakov
_ - podtržítko (podčiarovník), ktorý nahrádza iba jeden znak.

Predstavme si, že existuje akási kniha o **program**ovaní, ale nevieme jej presný názov. Vtedy použijeme príkaz:

```
mysql> select nazov, autor, vydavatel, cena from kniha
        where nazov like '%program%';
```

Výsledok je na výpise č.5-10:

```
mysql> select nazov, autor, vydavatel, cena
    -> from kniha
    -> where nazov like '%program%';
+-------------------------------+-------------------+-----------+--------+
| nazov                         | autor             | vydavatel | cena   |
+-------------------------------+-------------------+-----------+--------+
| Naucte se programovat v Delphi | Binzinger, Thomas | Grada     | 439.00 |
+-------------------------------+-------------------+-----------+--------+
```

Treba si uvedomiť, že v tomto ilustračnom príklade, kde máme iba sedem záznamov, je výsledok tohto hľadania pomerne jednoznačný. V praxi, kde sú v tabuľkách milióny záznamov, je nutné čo najpresnejšie popísať danú podmienku pre vyhľadávanie, aby sme dostali čo najkonkrétnejšie výsledky, inak by vyššie uvedenej podmienke mohli vyhovieť aj stovky knižných titulov.
Preto sa často používa aj znak _ podtržítko, ktoré nahráda iba jeden znak v podmienke.

Podmienka s matematickými operáciami

Práca s príkazom SELECT, kde je v podmienke matematická operácia sa neveľmi líši od práce s textovým reťazcom. Je tu iba jedna zmena - číselné hodnoty sa nezapisujú do apostrofov.
Pri vyhľadávaní s formulkou *WHERE* s číselnou hodnotou použijeme tento všeobecný zápis:

SELECT ***položka_1, položka_2, ...položka_n*** **FROM** ***meno_tabuľky***
WHERE ***názov_filtračného_stĺpca*** **OPERAND** ***číselná_hodnota***

Za operand môžeme dosadiť:
„=“ (rovná sa)
„<>“ nerovná sa
„<“ menší než
„>“ väčší než

Tu je niekoľko príkladov:

Ak chceme vyhľadať knihy, ktorých cena je rovná konkrétnemu číslu, použijeme zápis:

```
mysql> select nazov, autor, cena from kniha
        where cena = 66;
```

Výsledok je na výpise č.5-11:

```
mysql> select nazov, autor, cena
    -> from kniha
    -> where cena = 66;
+-------------+----------------------+-------+
| nazov       | autor                | cena  |
+-------------+----------------------+-------+
| Vtaky v trni | McCulloughova, Collen | 66.00 |
+-------------+----------------------+-------+
```

Znakom <> (nerovná sa) by sme analogicky vypísali všetky ostatné knihy, len nie tie, čo stáli 66 korún, tak ako je to na výpise č. 5-12:

```
mysql> select nazov, autor, cena
    -> from kniha
    -> where cena <>66;
+------------------------------+-----------------------+---------+
| nazov                        | autor                 | cena    |
+------------------------------+-----------------------+---------+
| Angelika a kral              | Golon, Anne a Serge   |   56.00 |
| KGB                          | Gordijevsky, Oleg     |  239.00 |
| Bratia Ricovci               | Simenon, Georges      |   18.00 |
| Linux - prakticky pruvodce   | Sobell, Mark G.       | 1073.00 |
| Naucte se programovat v Delphi | Binzinger, Thomas   |  439.00 |
| Pouzivame linux              | Welsh, M., Kaufman, L. |  494.00 |
+------------------------------+-----------------------+---------+
```

Taktiež môžeme stanoviť podmienku > (*väčší než*) alebo < (*menší než*).
Knihy drahšie ako 400 Sk sú na výpise č.5-13:

```
mysql> select nazov, autor, cena
    -> from kniha
    -> where cena > 400;
+------------------------------+-----------------------+---------+
| nazov                        | autor                 | cena    |
+------------------------------+-----------------------+---------+
| Linux - prakticky pruvodce   | Sobell, Mark G.       | 1073.00 |
| Naucte se programovat v Delphi | Binzinger, Thomas   |  439.00 |
| Pouzivame linux              | Welsh, M., Kaufman, L. |  494.00 |
+------------------------------+-----------------------+---------+
```

Knihy lacnejšie ako 400 Sk sú na výpise č.5-14:

```
mysql> select nazov, autor, cena
    -> from kniha
    -> where cena < 400;
+----------------+----------------------+--------+
| nazov          | autor                | cena   |
+----------------+----------------------+--------+
| Angelika a kral | Golon, Anne a Serge |  56.00 |
| KGB            | Gordijevsky, Oleg    | 239.00 |
| Bratia Ricovci | Simenon, Georges     |  18.00 |
| Vtaky v trni   | McCulloughova, Collen |  66.00 |
+----------------+----------------------+--------+
```

Podmienka s logiskými operáciami

Logické operácie v podmienkách príkazu SELECT sú tesne zviazané s vyššie uvedenými matematickými alebo textovými operáciami. Pomocou logických operandov môžeme spájať niekoľko podmienok do jedného príkazu. Obecný popis príkazu SELECT s využitím logických operácií vyzerá takto:

SELECT ***položka_1, položka_2, ...položka_n*** **FROM** ***meno_tabuľky***
WHERE ***podmienka_1*** **LOGICKÝ_OPERAND** ***podmienka_2*** **LOGICKÝ_OPERAND** ***podmienka_n***

Za logický operand môžeme použiť:
AND - a zároveň
OR - alebo
NOT (**!=**) - negácia = zápor

Záverom niekoľko príkladov:

AND

Ak hľadáme knihy, ktoré patria do kategórie odbornej literatúry a **ZÁROVEŇ** ich cena je väčšia ako 450 Sk, zadáme príkaz:

mysql> select nazov, autor, vydavatel from kniha
where cis_odd = 7 AND cena > 450;

```
mysql> select nazov, autor, vydavatel
    -> from kniha
    -> where cis_odd = 7 AND cena > 450;
+---------------------------+----------------------+----------------+
| nazov                     | autor                | vydavatel      |
+---------------------------+----------------------+----------------+
| Linux - prakticky pruvodce | Sobell, Mark G.      | Computer Press |
| Pouzivame linux           | Welsh, M., Kaufman, L. | Computer Press |
+---------------------------+----------------------+----------------+
```

Vidíme, že tejto zloženej podmienke vyhovujú dva záznamy.

OR

Ak hľadáme knihy, ktoré vydalo nakladateľstvo Computer Press **ALEBO** Grada, zadáme príkaz:

mysql> select nazov, autor, vydavatel from kniha
where vydavatel = 'Computer Press' OR vydavatel = 'Grada' ;

```
mysql> select nazov, autor, vydavatel
    -> from kniha
    -> where vydavatel ='Computer Press' OR vydavatel ='Grada';
+-------------------------------+------------------------+----------------+
| nazov                         | autor                  | vydavatel      |
+-------------------------------+------------------------+----------------+
| Linux - prakticky pruvodce    | Sobell, Mark G.        | Computer Press |
| Naucte se programovat v Delphi | Binzinger, Thomas      | Grada          |
| Pouzivame linux               | Welsh, M., Kaufman, L. | Computer Press |
+-------------------------------+------------------------+----------------+
```

Tentokrát vyhoveli podmienke tri záznamy.

!= (NOT)

Ak chceme nájsť knihy, ktoré NEVYDALO nakladateľstvo EAAP, zadáme príkaz:

mysql> select nazov, vydavatel, cena from kniha
where vydavatel != 'EAAP';

Výsledok je na výpise č.5-17:

```
mysql> select nazov, vydavatel, cena
    -> from kniha
    -> where vydavatel !='EAAP';
+-------------------------------+----------------------+---------+
| nazov                         | vydavatel            | cena    |
+-------------------------------+----------------------+---------+
| Angelika a kral               | Slovensky spisovatel |   56.00 |
| Bratia Ricovci                | Smena                |   18.00 |
| Vtaky v trni                  | Slovensky spisovatel |   66.00 |
| Linux - prakticky pruvodce    | Computer Press       | 1073.00 |
| Naucte se programovat v Delphi | Grada                |  439.00 |
| Pouzivame linux               | Computer Press       |  494.00 |
+-------------------------------+----------------------+---------+
```

Isto nás napadne, že je možné tieto typy podmienok vzájomne kombinovať, aby sme mohli vytvoriť aj veľmi zložitú podmienku.

Malé veľké databázy / 6.časť

V tejto časti seriálu budeme pokračovať vo vysvetľovaní príkazu **SELECT**. Minule sme si hovorili o základných agregačných funkciách ***SUM, MIN, MAX, COUNT, AVG*** a formulovali sme podmienku ***WHERE*** v tej najzákladnejšej forme.

Dnes si vysvetlíme a na príkladoch ukážeme ďalšie pokročilejšie parametre a príkazy. Budú to tieto kľúčové slová a klauzule: ***DISTINCT, BETWEEN, LIMIT, takzvané triedené výpisy a generované výpisy***. Aby sme však mali príklady ilustrovanejšie, doplníme si tabuľku **KNIHA** o ďalšie záznamy podľa priloženej tabuľky Tab.6-1:
(Samozrejme si môžete doplniť vlastné údaje)

id	nazov	autor	vydavatel	cis_odd	cena
8	Z polovnickej kapsy	Moric, Rudo	Mlade leta	4	89
9	Plebejska kosela	Mihalik, Vojtech	Slovensky spisovatel	1	15
10	Europou bez penazi	Hlubucek, Petr, Ing.	Roman Kasan	5	34

Ak sme tak vykonali (použitím príkazu INSERT INTO), pristúpime teraz k novým parametrom a kľúčovým slovám príkazu **SELECT**:

DISTINCT

Kľúčové slovo **DISTINCT** zamedzuje vypísaniu viacerých rovnakých hodnôt.
Ak nechceme, aby sa vypísali riadky, ktoré budú mať rovnaké hodnoty podľa stanovenej podmienky, použijeme toto kľúčové slovo. Potom sa takýto riadok vypíše iba raz. Predstavme si, že chceme vedieť, od akých vydavateľov máme v našej knižnici knihy. Môžeme použiť bežný príkaz **SELECT** s redukovaním stĺpcov na jeden, a to VYDAVATEL takto:

mysql> select vydavatel from kniha;

Dostaneme výsledok ako na výpise č. 6-1:

```
mysql> select vydavatel from kniha;
+----------------------+
| vydavatel            |
+----------------------+
| Slovensky spisovatel |
| EAAP                 |
| Smena                |
| Slovensky spisovatel |
| Computer Press       |
| Grada                |
| Computer Press       |
| Mlade leta           |
| Slovensky spisovatel |
| Roman Kasan          |
+----------------------+
10 rows in set (0.00 sec)
```

Vidíme, že príkaz **SELECT** vybral jeden stĺpec , ale z celej tabuľky, čím vypísal aj tie riadky, ktoré sa opakujú. Pri týchto pár riadkoch to nevadí, ale čo keď budeme mať tabuľku o tisícich záznamoch? Kto by v tom listoval? A práve na toto sa veľmi hodí **DISTINCT**. Takže použijeme :

mysql> select distinct vydavatel from kniha;

Vidíme, že výpis č. 6-2 je prehľadnejší, lebo zobrazil len sedem riadkov, čo sú všetci vydavatelia, OD ktorých knihy vlastníme.

```
mysql> select distinct vydavatel from kniha;
+----------------------+
| vydavatel            |
+----------------------+
| Computer Press       |
| EAAP                 |
| Grada                |
| Mlade leta           |
| Roman Kasan          |
| Slovensky spisovatel |
| Smena                |
+----------------------+
7 rows in set (0.05 sec)
```

BETWEEN

Parameter **BETWEEN** v podmienke ***WHERE*** určuje interval jej platnosti. Obecný zápis je :

SELECT ***meno_stĺpca*** **FROM** ***meno_tabuľky*** **WHERE** ***podmienka***
BETWEEN ***dolná_hranica*** **AND** ***horná_hranica***

Ak chceme vyhľadať knihy s minimálnou cenou napr. 200 Sk a maximálnou cenou 1000 Sk, urobíme to takto:

mysql> select id, nazov, autor, cena from kniha where cena BETWEEN 200 AND 1000;

Na výpise č.6-3 vidíme, že tejto podmienke vyhoveli tri záznamy.

```
mysql> select id, nazov, autor, cena from kniha
    -> where cena between 200 and 1000;
+----+--------------------------------+------------------------+--------+
| id | nazov                          | autor                  | cena   |
+----+--------------------------------+------------------------+--------+
|  2 | KGB                            | Gordijevsky, Oleg      | 239.00 |
|  6 | Naucte se programovat v Delphi | Binzinger, Thomas      | 439.00 |
|  7 | Pouzivame linux                | Welsh, M., Kaufman, L. | 494.00 |
+----+--------------------------------+------------------------+--------+
3 rows in set (0.06 sec)
```

(Musíme si uvedomiť, že táto podmienka sa porovnáva v stanovenom intervale od 200 do 1000, vrátane obidvoch krajných hodnôt. To len tak na okraj, lebo v matematike sa v určitých prípadoch krajné hodnoty intervalu neakceptujú.)

LIMIT

Ak sa domnievame, že prípadný výpis príkazu SELECT by bol veľmi dlhý a chceli by sme ho obmedziť na prvých n - riadkov, použijeme kľúčové slovo LIMIT na konci príkazu. Obecný zápis je :

SELECT ***mená_stĺcov*** **FROM** ***meno_tabuľky*** **WHERE** ***podmienka*** **LIMIT** ***n, m***

Ukážme si to teraz na našej tabuľke ZANER:

mysql> select * from zaner LIMIT 5;

Vidíme prvých päť riadkov výsledku príkazu SELECT, tak ako je to na výpise č. 6-4:

```
mysql> select * from zaner limit 5;
+---------+-------------+
| cis_odd | tematika    |
+---------+-------------+
|       1 | poezia      |
|       2 | roman       |
|       3 | krimi       |
|       4 | detska lit. |
|       5 | cestopis    |
+---------+-------------+
5 rows in set (0.44 sec)
```

Ak chceme vypísať ďalšie riadky z tabuľky, použijeme kľúčové slovo LIMIT s dvomi parametrami ***n*** a ***m***. Parameter **n** je *offset* a značí, od ktorého následujúceho riadku bude výpis pokračovať, a parameter **m** značí maximum vypísaných riadkov.
Takže príkaz :

mysql> select * from zaner LIMIT 5,10;

vypíše ďalších 10 riadkov od riadku, následujúcom po 5. riadku, teda riadky 6 až 15. Keďže naša tabuľka nemá toľko riadkov, výpis sa ukončí po poslednom riadku tabuľky, tak ako je to na výpise č.6-5:

```
mysql> select * from zaner limit 5,10;
+---------+--------------+
| cis_odd | tematika     |
+---------+--------------+
|       6 | lit. faktu   |
|       7 | odborna lit. |
+---------+--------------+
2 rows in set (0.00 sec)
```

Triedené výpisy

Výpisy príkazu **SELECT** môžeme formovať nielen pomocou kľúčových slov a agregačných funkcií, ale môžeme vytvárať aj tzv. ***triedené výpisy***.
Triedené výpisy tvoríme parametrami:

- ORDER BY (zoradenie vzostupne)
- ORDER BY DESC (zoradenie zostupne)
- GROUP BY (zoskupenie)
- HAVING (zoskupenie spĺňajúce podmienku)

ORDER BY

Príkazom **ORDER BY** zoradíme výpis podľa stanoveného stĺpca **vzostupne**.
Všeobecný zápis je:

SELECT ***názvy_stĺpcov*** **FROM** ***meno_tabuľky*** **ORDER BY** ***názvy_stĺpcov***

Ak chceme zoradiť výpis z tabuľky **KNIHA** podľa abecedy v stĺpci ***NAZOV***, zadáme príkaz:

mysql> select id, nazov, autor from kniha order by nazov;

tak ako je na výpise č. 6-6:

```
mysql> select id, nazov, autor from kniha
    -> order by nazov;
+----+--------------------------------+-----------------------+
| id | nazov                          | autor                 |
+----+--------------------------------+-----------------------+
|  1 | Angelika a kral                | Golon, Anne a Serge   |
|  3 | Bratia Ricovci                 | Simenon, Georges      |
| 10 | Europou bez penazi             | Hlubucek, Petr, Ing.  |
|  2 | KGB                            | Gordijevsky, Oleg     |
|  5 | Linux - prakticky pruvodce     | Sobell, Mark G.       |
|  6 | Naucte se programovat v Delphi | Binzinger, Thomas     |
|  9 | Plebejska kosela               | Mihalik, Vojtech      |
|  7 | Pouzivame linux                | Welsh, M., Kaufman, L.|
|  4 | Vtaky v trni                   | McCulloughova, Collen |
|  8 | Z polovnickej kapsy            | Moric, Rudo           |
+----+--------------------------------+-----------------------+
10 rows in set (0.05 sec)
```

Vidíme, že stĺpec ***NAZOV*** sa zoradil podľa abecedy, takže stĺpec ***ID*** už nejde po poradí.
Poznámka:
Musíme spomenúť, že práve tu vznikajú problémy s československým triedením. Ak nie je server nastavený na triedenie podľa českej abecedy, tak znaky s diakritikou nezaradí správne, teda A, Á, B, C, Č, D, Ď... atď. ale na koniec súboru v zmysle ASCII tabuľky.

Ak chceme použiť triedenie podľa viacerých stĺpcov tabuľky, zadáme názvy príslušných stĺpcov za ORDER BY. Vzorovým príkladom by mohlo byť zoradenie podľa názvu, potom podľa autorov a nakoniec podľa vydavateľstva. Zoradenie sa vykoná v tom poradí, v akom zadáme názvy jednotlivých stĺpcov.

ORDER BY DESC

Tento príkaz je veľmi podobný predchádzajúcemu, len slovko **DESC** znamená, že určený stĺpec sa zoradí **zostupne**. Ak teda chceme zoradiť výpis tabuľky **KNIHA** zostupne podľa stĺpca **AUTOR**, zadáme príkaz:

mysql> select id, nazov, autor from kniha order by autor desc;

Výsledok vidíme na výpise č. 6-7:

```
mysql> select id, nazov, autor from kniha
    -> order by autor desc;
+----+--------------------------------+-----------------------+
| id | nazov                          | autor                 |
+----+--------------------------------+-----------------------+
|  7 | Pouzivame linux                | Welsh, M., Kaufman, L.|
|  5 | Linux - prakticky pruvodce     | Sobell, Mark G.       |
|  3 | Bratia Ricovci                 | Simenon, Georges      |
|  8 | Z polovnickej kapsy            | Moric, Rudo           |
|  9 | Plebejska kosela               | Mihalik, Vojtech      |
|  4 | Vtaky v trni                   | McCulloughova, Collen |
| 10 | Europou bez penazi             | Hlubucek, Petr, Ing.  |
|  2 | KGB                            | Gordijevsky, Oleg     |
|  1 | Angelika a kral                | Golon, Anne a Serge   |
|  6 | Naucte se programovat v Delphi | Binzinger, Thomas     |
+----+--------------------------------+-----------------------+
10 rows in set (0.05 sec)
```

GROUP BY

Parametrom GROUP BY zoskupíme výsledok príkazu SELECT k stanovenému stĺpcu.
Obecný zápis je:

SELECT ***názvy_stĺpcov, agregačná_funkcia*** **FROM** ***meno_tabuľky*** **GROUP BY** ***názov_stĺpca_pre zoskupenie***

Predstavme si, že chceme spočítať sumu cien kníh po jednotlivých knižných oddelenia, napr. v oddelení č.1 je suma xy korún, v oddelení č.2 je suma yz korún a podobne.
Stačí, ak spočítame sumu cien a túto zoskupíme po oddeleniach.
Zadáme:

mysql> select cis_odd, sum(cena) as 'Celkom' from kniha group by cis_odd;

```
mysql> select cis_odd, sum(cena) as 'Celkom'
    -> from kniha group by cis_odd;
+---------+---------+
| cis_odd | Celkom  |
+---------+---------+
|       1 |   15.00 |
|       2 |  122.00 |
|       3 |   18.00 |
|       4 |   89.00 |
|       5 |   34.00 |
|       6 |  239.00 |
|       7 | 2006.00 |
+---------+---------+
7 rows in set (0.16 sec)
```

Na výpise č.6-8 vidíme sumy cien kníh po jednotlivých oddeleniach knižnice. Zároveň sme využili znalosti premenovania stĺpca agregačnej funkcie **SUM** na stĺpec *Celkom* pomocou ***AS***.

HAVING

Ak chceme vyššie uvedený príklad obmedziť určitou podmienkou, použijeme klauzulu HAVING.
Obecný zápis je:

SELECT ***názvy_stĺpcov, agregačná_funkcia*** **FROM** ***meno_tabuľky*** **GROUP BY** ***názov_stĺpca***
HAVING ***podmienka***

HAVING obmedzí rozsah výpisu tabuľky tým, že z agregovaných riadkov vyradí tie, ktoré nevyhovujú uvedenej podmienke.
Ak chceme výpis cien kníh po jednotlivých oddeleniach z predchádzajúceho príkladu obmedziť iba na tie riadky, kde suma je väčšia ako 100 (korún), zadáme príkaz:

mysql> select cis_odd, sum(cena) as 'Celkom' from kniha group by cis_odd having sum(cena)>100 ;

Na výpise č.6-9 vidíme, že je to redukovaný výpis č.6-8 a riadky 1,3,4 a 5 so sumou menšou ako 100 boli ignorované.

```
mysql> select cis_odd, sum(cena) as 'Celkom'
    -> from kniha group by cis_odd
    -> having sum(cena)>100;
+---------+---------+
| cis_odd | Celkom  |
+---------+---------+
|       2 |  122.00 |
|       6 |  239.00 |
|       7 | 2006.00 |
+---------+---------+
3 rows in set (0.11 sec)
```

Generované výpisy

Niekedy je veľmi potrebné uložiť obsah niektorej tabuľky (alebo aj celej databáze) do súboru a prípadné opätovné načítanie uložených dát do servera.
Môžeme tak konať z rôznych dôvodov. Pravidelná archivácia dát je jeden z najdôležitejších . Taktiež môžeme preniesť tieto dáta na iný počítač. (Veď kto by znova zadával údaje z klávesnice!).V takom prípade využijeme tzv. generované výpisy. Sú to také výpisy, kde príslušné príkazy SQL servera vygenerujú obsah stanovených databáz a ich tabuliek do súboru, ktoré je možné v prípade potreby spätne zo súboru načítať do servera.
Pomocných programov alebo príkazov je niekoľko, my sa naučíme používať štyri, rozdelené do týchto dvoch skupín:

a) príkazy, generujúce len obsah (dáta) tabuliek:

- **select .... into outfile** - vygeneruje dáta z tabuľky do súboru
- **load data infile** - načíta dáta zo súboru do tabuľky

b) príkazy, generujúce obsah aj formu (dáta aj štruktúru) tabuliek (alebo databáz):

- **mysqldump** - vygeneruje štruktúru aj dáta z tabuľky do súboru
- **source (\.)** programu mysql - vykoná SQL skript = načíta štruktúru aj dáta do SQL servera

SELECT ... INTO OUTFILE

Ak chceme uložiť iba dáta z ľubovolnej tabuľky do súboru, použijeme tento príkaz v monitore MySQL. Obecný tvar je :

SELECT * FROM *meno_tabuľky* INTO OUTFILE *'meno_súboru'* FIELDS TERMINATED BY *'znak'*

FIELDS TERMINATED BY znamená *polia ukončené* (znakom), tzv. oddelovačom.
Parameter ***znak*** je typ oddelovača jednotlivých stĺpcov. Spravidla to býva *čiarka* (,) alebo *pipe = rúra* (|).
Predstavme si, že chceme uložiť dáta z tabuľky **KNIHA** do súboru ***kniha.dat*** , kde oddelovačom stĺpcov bude pipe. Názov súboru nie je záväzný, prípona je ľubovoľná, ale **.dat** symbolizuje, že bude obsahovať dáta.
Vtedy zadáme:

mysql> select * from kniha into outfile 'kniha.dat' fields terminated by '|';

Keďže v našich dátach (v stĺpci AUTOR) už používame čiarku na oddelenie mena od priezviska, ako oddelovač použijeme iný znak, najvhodnejšie pipe, aby nedošlo k chybe.
Výpis č. 6-10 potvrdzuje, že všetko prebehlo v poriadku a bolo uložených 10 riadkov:

```
mysql> select * from kniha into outfile 'kniha.dat'
    -> fields terminated by '|';
Query OK, 10 rows affected (0.06 sec)
```

Vzhľadom na relatívnu cestu sa súbor ***kniha.dat*** nachádza v adresári *MYSQL\DATA\KNIZNICA*. Teraz sa môžeme pozrieť, aký je obsah súboru (výpis č.6-11) :

```
1|Angelika a kral|Golon, Anne a Serge|Slovensky spisovatel|2|56.00|
2|KGB|Gordijevsky, Oleg|EAAP|6|239.00|
3|Bratia Ricovci|Simenon, Georges|Smena|3|18.00|
4|Vtaky v trni|McCulloughova, Collen|Slovensky spisovatel|2|66.00|
5|Linux - prakticky pruvodce|Sobell, Mark G.|Computer Press|7|1073.00|
6|Naucte se programovat v Delphi|Binzinger, Thomas|Grada|7|439.00|
7|Pouzivame linux|Welsh, M., Kaufman, L.|Computer Press|7|494.00|
8|Z polovnickej kapsy|Moric, Rudo|Mlade leta|4|89.00|
9|Plebejska kosela|Mihalik, Vojtech|Slovensky spisovatel|1|15.00|
10|Europou bez penazi|Hlubucek, Petr, Ing.|Roman Kasan|5|34.00|
```

Je zrejmé, že oddelovač pipe (|) oddeľuje jednotlivé stĺpce tabuľky.

Teraz si môžeme tento súbor odložiť na bezpečné miesto, aby sme ho použili, keď to bude potrebné.

LOAD DATA INFILE

Predstavme si situáciu, že z určitého dôvodu je zrazu tabuľka **KNIHA** prázdna. To sa môže stať pri nechcenom výmaze (vyprázdnení) celej tabuľky (*delete from kniha* - spomínate si?), alebo je nekorektná, chýbaju jej niektoré záznamy a tak sme ju vyprázdnili. Ak máme odložený aktuálny! súbor ***kniha.dat***, nemusíme byť mrzutí. Stačí, ak použijeme príkaz **LOAD DATA INFILE**.
Jeho obecný tvar je:

LOAD DATA INFILE *meno_súboru* INTO TABLE *meno_tabuľky* FIELDS TERMINATED BY *'znak'*

Takže v našom prípade skopírujeme súbor do príslušného adresára (s názvom databáze) a zadáme príkaz:

mysql> load data infile 'kniha.dat' into table kniha fields terminated by '|';

Znak, ktorým sú oddelené jednotlivé stĺpce, zistíme nahliadnutím do súboru.
Výpis č.6-12 potvrdzuje, že bolo nahratých desať riadkov do tabuľky:

```
mysql> load data infile 'kniha.dat' into table kniha
    -> fields terminated by '|';
Query OK, 10 rows affected (0.33 sec)
Records: 10  Deleted: 0  Skipped: 0  Warnings: 0
```

Obsah súboru ***kniha.dat*** môžeme nahrať aj do prázdnej tabuľky s iným názvom ako je **KNIHA**. Podmienkou je, že táto prázdna tabuľka musí existovať a musí mať presne tú istú štruktúru ako tabuľka **KNIHA!**

MySQLDump

Ak chceme uložiť do súboru nielen dáta, ale aj štruktúru určitej tabuľky alebo aj celej databáze, je lepšie použiť program **MySQLDump**. Pozor! Nie je to príkaz monitoru MySQL, ale je to samostatný pomocný program MySQL servera. Preto sa spustí v príkazovom riadku príslušného operačného systému.
Obecný zápis je:

mysqldump *meno_databáze meno_tabuľky* > *meno_súboru*

Zobáčik „**>**" znamená, že výstup programu *mysqldump* sa presmeruje do súboru ***meno_súboru.*** Ak by sme ho nezadali, výstup programu by sa smeroval štandardne na obrazovku. Ak by sme zadali za zobáčikom namiesto mena súboru slovko **prn**, výstup by sa presmeroval na pripojenú tlačiareň.
Pozrime sa bližšie na obsah súboru ***zaner.sql*** (výpis č. 6-13):

```
# MySQL dump 8.2
#
# Host: localhost    Database: kniznica
#--------------------------------------------------------
# Server version  3.22.34-shareware-debug

#
# Table structure for table 'zaner'
#

CREATE TABLE zaner (
  cis_odd int(11) DEFAULT '0' NOT NULL auto_increment,
  tematika varchar(20),
  PRIMARY KEY (cis_odd)
);

#
# Dumping data for table 'zaner'
#

INSERT INTO zaner VALUES (1,'poezia');
INSERT INTO zaner VALUES (2,'roman');
INSERT INTO zaner VALUES (3,'krimi');
INSERT INTO zaner VALUES (4,'detska lit.');
INSERT INTO zaner VALUES (5,'cestopis');
INSERT INTO zaner VALUES (6,'lit. faktu');
INSERT INTO zaner VALUES (7,'odborna lit.');
```

Z výpisu vidíme, že súbor ***zaner.sql*** obsahuje SQL príkazy pre server. Na začiatku sú SQL príkazy pre vytvorenie tabuľky **ZANER** (*create table*), následujú SQL príkazy pre naplnenie tabuľky príslušnými dátami (*insert into*).
Takto vygenerovaný výpis je veľmi užitočný. Je zrejmé, že ho prípadne môžeme upraviť, doplniť dáta alebo prepracovať štruktúru tabuľky. A hlavne ho môžeme použiť aj tam, kde nie je nadefinovaná príslušná tabuľka.

Takto generovaný výpis je veľmi vhodný na prenášanie celých databáz na iný počítač so serverom MySQL. A teraz už chápete, prečo som mu dal príponu **.sql**, aj keď som ho mohol pomenovať ľubovoľne.

Source (\.) programu MySQL

Môžeme povedať, že súbor ***zaner.sql*** je akýsi dávkový súbor SQL príkazov. Hovoríme mu aj **SQL skript** (z angl. script). No dobre, ale ako najjednoduchšie tieto príkazy vykonať bez toho, aby sme ich pracne preťukávali zo súboru do klávesnice? No predsa jednoduchým spustením tohto skriptu. Náš starý známy monitor MySQL má jeden parameter, ktorý umožňuje vykonať súbor s SQL skriptom automaticky tak, ako keby boli jednotlivé príkazy zadávané z klávesnice. Tento parameter sa nazýva **source** (zdroj), a označuje sa aj skrátene „**\.**" (pozri help). Ak teda zadáme:

mysql> source zaner.sql;

dostaneme požadovaný výsledok. Výpis č.6-14 ukazuje, že boli vykonané jednotlivé príkazy zo súboru ***zaner.sql***:

```
mysql> source zaner.sql
Query OK, 0 rows affected (0.22 sec)

Query OK, 1 row affected (0.22 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

Že sa tak skutočne stalo, môžeme sa presvedčiť už známym príkazom ***select * from zaner***.

Nabudúce si vysvetlíme vzájomné spojovanie tabuliek a manipuláciu s časovými údajmi SQL servera.

Malé veľké databázy / 7.časť

V minulej časti seriálu sme si ukázali, ako vytvoríme súbor zaner.sql. Vieme, že tento súbor obsahuje SQL príkazy pre vytvorenie a zároveň naplnenie požadovanej tabuľky.

Source (\.) programu MySQL
Môžeme povedať, že súbor ***zaner.sql*** je akýsi dávkový súbor SQL príkazov. Hovoríme mu aj **SQL skript** (z angl. script). No dobre, ale ako najjednoduchšie tieto príkazy vykonať bez toho, aby sme ich pracne preťukávali zo súboru do klávesnice? No predsa jednoduchým spustením tohto skriptu. Náš starý známy monitor MySQL má jeden parameter, ktorý umožňuje vykonať súbor s SQL skriptom automaticky tak, ako keby boli jednotlivé príkazy zadávané z klávesnice. Tento parameter sa nazýva **source** (zdroj), a označuje sa aj skrátene „**\.**" (pozri help). Ak teda zadáme:

mysql> source zaner.sql;

dostaneme požadovaný výsledok. Výpis č.6-14 ukazuje, že boli vykonané jednotlivé príkazy zo súboru ***zaner.sql***:

```
mysql> source zaner.sql
Query OK, 0 rows affected (0.22 sec)

Query OK, 1 row affected (0.22 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

Veľmi často potrebujeme v databázach okrem textových a číselných hodnôt uchovávať aj údaje o čase a dátume. Tieto zaznamenávajú rôzne udalosti v konkrétnom okamihu. Asi najdôležitejším údajom býva dátum. Ako príklad môže poslúžiť príklad, keď sa v každej knižnici zaznamenáva dátum zápožičky a dátum vrátenia kníh. Aj o každom úradnom úkone sa zaznamenáva dátum jeho vykonania. Preto si dnes povieme niečo o uchovávaní dátumu a času a ukážeme si aj operácie s týmito údajmi.

Všetko to, čo sa týka dátumu a času môžeme v MySQL rozdeliť do dvoch skupín:
- *dátumové a časové typy*
- *dátumové a časové funkcie*

Najdôležitejšie dátové typy už poznáme. Sú to buď číselné typy (napr. INT, ...), alebo textové typy (CHAR, VARCHAR, ...). Teraz k nim pridáme dátumové a časové typy.

Dátumové a časové typy

MySQL podporuje tieto časové a dátumové typy:
TIME - uloženie časových údajov
DATE - uloženie dátumu
DATETIME - združené údaje dátumu a času
YEAR - uloženie roku
TIMESTAMP - tzv. časové razítko

Formát jednotlivých typov sa líši, preto si teraz tieto typy preberieme:

DATETIME

Tento typ sa používa, keď potrebujeme hodnotu, ktorá obsahuje obidve informácie - dátumovú aj časovú.

Formát: **‘RRRR-MM-DD HH:MM:SS’**
Podporovaný rozsah: od ‘1000-01-01 00:00:00’ do ‘9999-12-31 23:59:59’
Vidíme, že podporovaný rozsah je veľmi veľký, ktorý vyhovie aj veľmi náročným aplikáciám.

DATE
Tento typ použijeme, keď potrebujeme iba dátumovú informáciu, bez časovej časti.

Formát: **‘RRRR-MM-DD’**
Podporovaný rozsah: od ‘1000-01-01’ do ‘9999-12-31’

TIMESTAMP
Tento typ je veľmi zvláštny. Pomenujeme si ho aj časové razítko. Skutočne, ak máme v tabuľke nadefinovaný určitý stĺpec typu **TIMESTAMP**, vždy a automaticky sa pri operácii ***INSERT*** alebo ***UPDATE*** na stanovenom riadku zároveň do tejto položky uloží aktuálny časový údaj v stanovenom formáte.

Formát: **‘RRRRMMDDHHMMSS’**
Podporovaný rozsah: od ‘19700101000000’ do ‘20371231235959’
Všimnime si podporovaný rozsah. Začína počiatkom roku 1970 a končí poslednou sekundou v roku 2037. Ale, myslím, že aj tento rozsah postačuje našej práci.
Tento typ môže mať aj zmenený rozsah. Stačí, ak pri jeho definovaní stanovíme parametrom príslušnú veľkosť zaznamenávaného poľa tak, ako je to v tab. č.7-1:

Parameter	Zobrazovaný formát
TIMESTAMP(14)	RRRRMMDDHHMMSS
TIMESTAMP(12)	RRMMDDHHMMSS
TIMESTAMP(10)	RRMMDDHHMM
TIMESTAMP(8)	RRRRMMDD
TIMESTAMP(6)	RRMMDD
TIMESTAMP(4)	RRMM
TIMESTAMP(2)	RR

Ak si pozorne prezrieme túto tabuľku, základný formát sa neskracuje zprava, ale mení sa formát zobrazovania roku zo štvorciferného na dvojciferný. Treba pripomenúť, že ak zadáme typ TIMESTAMP bez parametra, defaultne (štandardne) sa uvažuje plný 14-miestny formát.

Poznámka:
Aj keď sú **DATE**, **DATETIME** a **TIMESTAMP** veľmi príbuzné, nemajú rovnaký rozsah platnosti. Zatiaľ čo dátum 20-11-1963 môže byť uložený v **DATE** alebo **DATETIME**, pre **TIMESTAMP** je neplatný a bude konvertovaný na nuly.
Ako by sme mohli túto zaujímavú funkciu využiť? To si ukážeme nižšie.

TIME
Tento typ je zobrazovaný vo formáte **HH:MM:SS**, (alebo **HHH:MM:SS** pre rozšírenú hodnotu) a môže dosahovať rozsah hodnôt od -838:59:59 do 838:59:59. Nie, nie je to preklep, v tomto type môžeme ukladať aj časový údaj väčší ako 24 hodín, alebo dokonca môže nadobúdať záporné hodnoty.
‘**TIME**’ hodnoty môžeme zadávať v týchto formátoch:

a) ako reťazec v **‘HH:MM:SS’** formáte
   Sú možné aj príbuzné syntaxie, teda ‘10:24:56’ je rovnaký ako ‘10.24.56’
b) ako reťazec bez dvojbodiek, teda vo formáte **‘HHMMSS’**
   Teda ‘101112’ je chápaný ako ‘10:11:12’. Ale zápis ‘109745’ je neplatný, lebo 97 minút neexistuje
c) ako číslo vo **HHMMSS** formáte (teda bez apostrofov)
   Číslom 101112 sa rozumie čas 10:11:12
d) ako výsledok funkcie, ktorá vráti hodnotu, akceptovateľnú **TIME** kontextom, napr. ***Current_Time***

Pre **TIME** hodnoty, prezentované ako reťazec, nie je nutné používať dvoch cifier v prezentácii hodín, minút a sekúnd, ktoré sú menšie ako číslo 10. Teda zápis ‘8:3:2’ je to isté ako ‘08:03:02’.

Pozor!
Buďme opatrní pri zadávaní neúplných údajov! MySQL “počíta” číslice zprava. Predpokladá, že dve pravé číslice reprezentujú sekundy. Takže zápis ‘11:12’, ‘1112’ a 1112 (spomeňme si, že v apostrofoch sú reťazce - strings, bez apostrofov čísla - numeric) reprezentuje 00:11:12, teda 11 minút a 12 sekúnd, a nie 11 hodín a 12 minút! Jednoducho - ‘12’ prezentuje 12 sekúnd.
Ak by sme chceli zadať údaj vačší ako povolený rozsah, napr. 850:00:00, bude aproximovaný na maximálny rozsah, teda 838:59:59. Neplatné časové údaje sú konvertované na 00:00:00, takže je veľmi ťažké rozhodnúť, či sa do programu uložil čas polnoci, alebo niekto zadal zlý časový údaj, ktorý bol na tento konvertovaný.

YEAR

YEAR je jednobajtový typ, ktorý reprezentuje roky. MySQL vracia a zobrazuje **YEAR** hodnoty vo formáte **‘YYYY’**. Rozsah je od ‘1901’ ‘2155’.
YEAR hodnoty môžeme zadávať v týchto formátoch:

a) ako 4-miestny reťazec v rozsahu od ‘1901’ do ‘2155’
b) ako 4-miestne číslo v rozsahu od 1901 do 2155
c) ako 2-miestny reťazec v rozsahu od ‘00’ do ‘99’.
   Hodnoty v rozsahu ‘00’ až ‘69’ sú konvertované na hodnoty od ‘2000’ do ‘2069’, hodnoty v rozsahu ‘70’ až ‘99’ sú konvertované na hodnoty od ‘1970’ do ‘1999’
d) ako 2-miestne číslo v rozsahu 1 až 99.
   Hodnoty v rozsahu 1 až 69 sú konvertované na hodnoty od 2001 do 2069, hodnoty v rozsahu 70 až 99 sú konvertované na hodnoty od 1970 do 1999. Všimnume si, že v tomto prípade nemôžeme zadefinovať ten magický rok 2000! Preto doporučujem používať rok v reťazcovom formáte.
e) ako výsledok funkcie, ktorá vracia hodnoty akceptovateľné v **YEAR** kontexte, napr. ***NOW()***.

Uvedomme si, že nekorektné hodnoty **YEAR** sú konvertované na hodnotu ‘0000’ .

Dátumové a časové funkcie

Zatiaľ čo dátumové a časové typy určovali štruktúru - formát, v akej sú ukladané dané hodnoty, dátumové a časové funkcie vracajú práve tie hodnoty. Tak ako ozajstné funcie, aj tieto spravidla obsahujú parameter, na základe ktorého vracajú požadovanú hodnotu.
Teraz si ukážeme najdôležitejšie funkcie, ktoré asi najčastejšie použijeme v našich projektoch.
A ako vyzískame tieto hodnoty? No predsa pomocou nám už veľmi známeho príkazu SQL - ***SELECT***. Názvy jednotlivých funkcií sú odvedené od ich činností v anglickom jazyku. Pre lepšiu čitateľnosť ich budeme uvádzať s veľkými prvými písmenami. Zadávať ich však môžeme rôzne. Presné použitie pri jednotlivých ukážkach je zrejmé z príslušných výpisov:

DayOfWeek(*dátum*)

Vracia index dňa v týždni v zadanom dátume. Index dňa je číslená hodnota, kde
1 = Sunday, 2 = Monday, ..., 7 = Saturday. Na výpis č.7-1 vidíme, že 20.januára tohto roku bola naozaj sobota.

```
mysql> select dayofweek('2001-01-20');
+--------------------------+
| dayofweek('2001-01-20') |
+--------------------------+
|                        7 |
+--------------------------+
```

WeekDay(*dátum*)

Je veľmi podobný ako ***DayOfWeek***. Vracia index dňa v týždni, ale index nadobúda tieto hodnoty: 0 = Monday, 1 = Tuesday, ... , 6 = Sunday. Všimnime si, že zatiaľ čo v prvom prípade sme dátum zadali ako reťazec (string), v tomto prípade sme použili numerický zápis. To potvrdzuje vyššie spomenuté formáty ukladania časových a dátumových typov.

```
mysql> select weekday(20010120);
+------------------+
| weekday(20010120) |
+------------------+
|                5 |
+------------------+
```

DayOfMonth(*dátum*)

Ak chceme z dátumu vyselektovať číslo dňa v danom mesiaci, použijeme ***DayOfMonth***, tak ako na výpise 7-3. Aj tu sme použili jeden z možných variantov zápisu dátumu:

```
mysql> select dayofmonth('20010120');
+-----------------------+
| dayofmonth('20010120') |
+-----------------------+
|                    20 |
+-----------------------+
```

DayOfYear(*dátum*)

Koľkýže to bude deň od začiatku roku, ten 3.máj 2001? No predsa stodvadsiatytretí, tak ako na výpise č.7-4:

```
mysql> select dayofyear('2001-05-03');
+-------------------------+
| dayofyear('2001-05-03') |
+-------------------------+
|                     123 |
+-------------------------+
```

DayName(*dátum*)

Ak nám nevyhovuje, že funkcie ***DayOfWeek*** alebo ***WeekDay*** vracajú index dňa v týždni a my by sme radšej prijali názov dňa (bohužiaľ len v angličtine), použijeme funkciu DayName tak, ako na výpise č.7-5:

```
mysql> select dayname(20010120);
+------------------+
| dayname(20010120) |
+------------------+
| Saturday         |
+------------------+
```

MonthName(*dátum*)

Obdobným spôsobom môžeme získať názov mesiaca z dátumu. Použijeme funkciu ***MonthName*** tak, ako na výpise č.7-6:

```
mysql> select monthname(20010120);
+--------------------+
| monthname(20010120) |
+--------------------+
| January            |
+--------------------+
```

Year(*dátum*)

Tak, ako sme selektovali den v mesiaci, je možné selektovať aj rok. Príklad je na výpise č.7-7:

```
mysql> select year('010120');
+---------------+
| year('010120') |
+---------------+
|          2001 |
+---------------+
```

Všimnime si, že sme rok zadali len v dvojmiestnom tvare, ale SQL server ho reprezentuje v plnom štvormiestnom výpise. Pozor! Nemýľme si meno funkcie z názvom dátového typu!

Hour(*čas*)

Touto funkciou vyselektujeme hodinu zo zadaného časového údaju (výpis č.7-8):

```
mysql> select hour('15:23:56');
+------------------+
| hour('15:23:56') |
+------------------+
|               15 |
+------------------+
```

Minute(*čas*)

Aj minúty môžeme vyselektovať zo zadaného času (výpis č.7-9):

```
mysql> select minute(184556);
+----------------+
| minute(184556) |
+----------------+
|             45 |
+----------------+
```

Všimnime si formát zadávania času (výpis č.7-10) ako aj v ˇďaľších príkladoch:

```
mysql> select minute('14.26.45');
+--------------------+
| minute('14.26.45') |
+--------------------+
|                 26 |
+--------------------+
```

Second(*čas*)

Obdobne je to so sekundami, ale teraz zadáme čas ako číslo (výpis č.7-11):

```
mysql> select second(142839);
+----------------+
| second(142839) |
+----------------+
|             39 |
+----------------+
```

Predstavme si, že nedopatrením zadáme nevhodný formát času, napr. 12 hodín 25 minút a 79 sekúnd. Čo sa stane? Odpoveď je na výpise č.7-12:

```
mysql> select second('12:25:79');
+--------------------+
| second('12:25:79') |
+--------------------+
|               NULL |
+--------------------+
```

Je to prázdna hodnota. (Pozor! Nie nulová!)

To_Days(*dátum*)

Táto funkcia vráti počet dní od roku 0 (nula). Príklad je na výpise č.7-13:

```
mysql>  select to_days(20010120);
+--------------------+
| to_days(20010120) |
+--------------------+
|             730870 |
+--------------------+
```

Áno, je to presne 730870 dní od počiatku letopočtu.

From_Days(*n*)

Naopak, ak vieme, koľko dní od počiatku letopočtu to bolo a my chceme vedieť presný dátum, použijeme túto funkciu tak, ako na výpise č.7-14:

```
mysql> select from_days(730871);
+--------------------+
| from_days(730871) |
+--------------------+
| 2001-01-21         |
+--------------------+
```

Nie, nepočítal som to, len som predchádzajúci príklad zväčšil o jeden deň. A funguje to!

Date_Format(*dátum, formát*)

Niektoré vyššie spomenuté selekčné funkcie sú integrované v tejto veľmi zaujímavej funkcii. Pomocou nej sme schopní vytvoriť rozmanité výpisy. Táto funkcia obsahuje dva parametre: dátum a formát. Parameter formát určuje, ako bude formulovaný výsledný výpis. Príklady najčastejšie používaných formátov sú v Tab.č.7-2:

Formát	Popis
%M	meno mesiaca napr. January, ...December
%W	deň v tyždni napr. Sunday ...Saturday
%Y	rok, na 4 číslice
%y	rok, na 2 číslice
%a	skrátený deň v týždni napr. Sun ...Sat
%b	skrátené meno mesiaca napr. Jan...Dec
%d	deň v mesiaci, číselne 00...31
%e	deň v mesiaci, číselne 0...31
%m	mesiac, číselne 01...12
%c	mesiac, číselne 1...12
%j	deň v roku 001...366
%H	hodiny 00...23
%k	hodiny 0...23
%h	hodiny 01...12
%l	hodiny 1...12
%i	minúty 00...59
%S	sekundy 00...59
%r	čas 12-hodinový formát hh:mm:ss AM/PM
%T	čas 24-hodinový formát hh:mm:ss

Ak chceme zo zadaného dátumu a času vyformátovať zápis, kde bude názov dňa, názov mesiaca a rok, použijeme parametre %W, %M a %Y tak, ako na výpise č.7-15:

```
mysql> select date_format('2001-01-20 15:25:46', '%W %M %Y');
+-------------------------------------------------+
| date_format('2001-01-20 15:25:46', '%W %M %Y') |
+-------------------------------------------------+
| Saturday January 2001                           |
+-------------------------------------------------+
```

Efektné, však?!?

Time_Format(*čas, formát*)

Toto je obdoba predchádzajúcej funkcie, ale so zameraním na časovú zložku. Preto môžeme používať len tie parametre, ktoré sa týkajú času. Ostatné budú poskytovať hodnotu ***NULL*** alebo ***0*** (nula).

CurDate(), Current_Date()

Táto funkcia bez parametrov vráti aktuálny dátum v operačnom systéme. Príklad je na výpise č.7-16:

```
mysql> select curdate();
+------------+
| curdate()  |
+------------+
| 2001-01-20 |
+------------+
```

Ak by sme potrebovali dostať výsledok v numerickom tvare, stačí ak v príkaze SELECT pripočítame nulu, tak ako je to na výpise č.7-17:

```
mysql> select curdate() + 0;
+---------------+
| curdate() + 0 |
+---------------+
|      20010120 |
+---------------+
```

CurTime(), Current_Time()

Táto funkcia bez parametrov vráti aktuálny čas v operačnom systéme. Príklad je na výpise č.7-18:

```
mysql> select curtime();
+-----------+
| curtime() |
+-----------+
| 15:51:23  |
+-----------+
```

Ak by sme potrebovali dostať výsledok v numerickom tvare, stačí ak v príkaze SELECT pripočítame nulu, tak ako je to na výpise č.7-19:

```
mysql> select curtime() + 0;
+---------------+
| curtime() + 0 |
+---------------+
|        162537 |
+---------------+
```

Now(), SysDate(), Current_TimeStamp()

Tak túto funkciu mám najradšej. Používam ju veľmi často. Prečo? Lebo spája obidve predchádzajúce funkcie. Výpis č.7-20 ukazuje, čo je hodnotou tejto funkcie:

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2001-01-20 15:54:43 |
+---------------------+
```

Aj tento výsledok môžeme pričítaním nuly previesť na numerický tvar (výpis č.7-21):

```
mysql> select now() + 0;
+----------------+
| now() + 0      |
+----------------+
| 20010120162947 |
+----------------+
```

Napadlo vás, ako by sa efektne dala táto fukcia využiť? No predsa môžeme túto funkciu použiť ako parameter dátum alebo čas vo vyššie uvedených parametrických funkciách! Takisto môžeme použiť všetky bezparametrické funkcie. Presvedčme sa jedným príkladom na výpise č.7-22:

```
mysql> select date_format(now(), '%W %M %Y %T');
+-----------------------------------+
| date_format(now(), '%W %M %Y %T') |
+-----------------------------------+
| Saturday January 2001 16:02:30    |
+-----------------------------------+
```

Jednoduché, nie?
Túto funkciu často využijeme pri zadávaní časového razítka, tak ako si ukážeme neskôr.

Sec_To_Time(*sekundy*)

Táto funkcia prevedie zadaný počet sekúnd na čas vo formáte HH:MM:SS, tak ako na výpise č.7-23:

```
mysql> select sec_to_time(3659);
+-------------------+
| sec_to_time(3659) |
+-------------------+
| 01:00:59          |
+-------------------+
```

Ak neveríte, prepočítajte si to za domácu úlohu.

Time_To_Sec(*čas*)

Táto funkcia je opakom predchádzajúcej. Zo zadaného času vypočíta dĺžku trvania v sekundách. Príklad je na výpise č.7-24:

```
mysql> select time_to_sec(curtime());
+------------------------+
| time_to_sec(curtime()) |
+------------------------+
|                  58300 |
+------------------------+
```

Aj v tomto príklade sme čas nezadali ako reťazec, ale sme využili bezparametrovú funkciu curtime() ako parameter parametrickej funkcie ***time_to_sec***.

Použitie

Vidíme, že MySQL má mnoho funkcií na prácu s časom a dátumom. Ich využitie je veľmi široké. Asi najviac využijeme funkcie na získanie týchto údajov v základnom formáte. Ale keď budeme potrebovať vypočítať rozdiel časov alebo dátumov, použijeme funkcie na prevod na počet dní, tie potom odpočítame a výsledok prevedieme späť na konkrétny dátum.

Cvičenie
Aby sme si ujasnili, ako pracujú tieto funkcie pri vkladaní do tabuľky, vytvorme si cvičnú tabuľku s názvom **CASOMIERA** s týmito stĺpcami:

Položka	Typ položky
cas	TIME
datum	DATE
datcas	DATETIME
rok	YEAR
razitko	TIMESTAMP

Teraz vložíme pomocou príkazu ***INSERT INTO*** postupne tieto hodnoty:

mysql> insert into casomiera values (curtime(), curdate(), now(), year(now()), null);

mysql> insert into casomiera values (‘’,’’,’’,’’,’’);

mysql> insert into casomiera values (null, null, null, null, null);

mysql> insert into casomiera(cas) values (curtime());

Takto sme naplnili tabuľku **CASOMIERA**, v ktorej sú štyri záznamy.

Teraz vykonáme príkaz :

mysql> select * from casomiera;

a dostamene výsledok ako na výpise č.7-25:

```
mysql> select * from casomiera;
+----------+------------+---------------------+------+----------------+
| cas      | datum      | datcas              | rok  | razitko        |
+----------+------------+---------------------+------+----------------+
| 17:03:13 | 2001-01-20 | 2001-01-20 17:03:13 | 2001 | 20010120170313 |
| 00:00:00 | 0000-00-00 | 0000-00-00 00:00:00 | 2000 | 00000000000000 |
| NULL     | NULL       | NULL                | NULL | 20010120170630 |
| 17:21:34 | NULL       | NULL                | NULL | 20010120172134 |
+----------+------------+---------------------+------+----------------+
4 rows in set (0.05 sec)
```

Pozrime sa bližšie na jednotlivé záznamy. Vieme, že jednotlivé riadky tabuľky **CASOMIERA** zodpovedajú postupnému zadávaniu vyššie uvedeného príkazu ***INSERT***.

Všimnime si tieto dôležité skutočnosti:

- Aj keď sme v 1. riadku zadali hodnotu ***null*** pre položku **RAZITKO**, predsa sa uložil aktuálny dátum a čas operačného systému. Ako keby SQL server orazítkoval (dátumom a časom) to, že vykonal tento príkaz. Porovnajme, že sa obsah stĺpca **RAZITKO** zhoduje s údajmi v stĺpcoch **CAS, DATUM a DATCAS**.
- Ak sme príkazom ***INSERT*** vkladali do všetkých položiek prázdne reťazce (‘’), SQL server zkonvertoval tieto položky na samé nuly, vrátane časového razítka **TIMESTAMP**.
- Ak sme vložili hodnotu ***null*** (čo nie je nula, ani prázdny retazec, ale akési prázdno), všetky položky nadobudli hodnotu ***null***, okrem časového razítka. Takto zase server orazil vykonanie tohoto príkazu.
- Nakoniec sme vložili iba hodnotu aktuálneho času systému (***curtime***()) do položky **CAS**. Ako vidíme, táto sa zapísala, ostatné položky nadobudli hodnotu ***null***. A znova server orazil vykonanie tohoto príkazu.

Je zrejmé, že z údajov časového razítka sme schopní vyčítať, kedy bol daný záznam uložený do tabuľky. Pripomínam, že sa **TIMESTAMP**, teda razítko mení nielen pri príkaze ***INSERT***, ale aj pri príkaze ***UPDATE***. Ostatné SQL príkazy nemajú na časové razítko žiadny vplyv. Vyskúšajte!

Tááák, týmto sme prebrali najdôležitejšie SQL príkazy a parametre príkazu ***SELECT***. Ich počet je podstatne rozsiahlejší, ale pre našu prácu bude stačiť tento výčet. Dnes sa mi nevyšiel priestor, aby sme si ukázali jednu z najdôležitejších činností SQL servera - spájanie tabuliek. Takže nabudúce.

Malé veľke databázy /8.časť

Predchádzajúce časti sme sa pri vysvetľovaní príkazu ***SELECT*** zaoberali iba jednou tabuľkou. Toto sa však v praxi používa zriedka. Veľmi často sa výstupné zostavy skladajú zo spojenia dvoch alebo aj viacerých tabuliek. Spojiť tabuľky potrebujeme vtedy, ak v jednej tabuľke nie sú všetky nami požadované informácie. A tak sa dnes budeme venovať spojovaniu tabuliek, ktorému sa anglicky hovorí **JOIN**.

Vytvorenie cvičných tabuliek

Aby sme si vysvetlili túto tématiku, vytvoríme si dve pomocné tabuľky, na ktorých budeme spojovanie tabuliek cvičiť. Prvá tabuľka sa nazýva **CITATEL**, v ktorej sú uložené cvičné mená jednotlivých čitatelov a identifikačné číslo vypožičanej literatúry. Druhá tabuľka nesie názov **LITERATURA** a obsahuje identifikačné číslo a názov literatúry, ktorú si môžu jednotliví čitatelia vypožičať. Z minula vieme, že nie je veľmi vhodné dávať do jednej tabuľky meno čitateľa a plný názov knihy, ktorú si požičal, napr. Hrbatý, Teória relativity. Veď kto by stále vypisoval celé názvy u všetkých čitateľov, keď je podstatne jednoduchšie zapísať iba identifikačné číslo príslušnej literatúry. (**Pozn**. Okrem toho by takto zostavená databáza odporovala zásadám návrhu podľa tzv. „normálnych foriem“, čo sme si ešte nevysvetľovali. Aj naša doteraz používaná tabuľka **KNIHA** čiastočne odporuje týmto princípom, a preto ju budeme neskôr upravovať. Zatiaľ ale na vysvetlenie základov SQL plne vyhovuje.)

Tab.č.8-1: CITATEL

meno	id_lit
Novak	1
Hrbaty	2
Janik	NULL
Bartak	4
Horakova	1

Tab.č.8-2: LITERATURA

id_lit	nazov
NULL	Sipkova Ruzenka
1	Amaterske radio
2	Teoria relativity
3	Cestovny poriadok ZSR

Všimnime si, že v prvej tabuľke je pri mene *Janik* prázdna hodnota (null). To znamená, že pán Janík nemá v tomto okamžiku vypožičanú žiadnu knihu. Podobne, v druhej tabuľke je zaznamenané, že kniha o Šípkovej Ruženke nemá ešte pridelené výpožičkové číslo, napr. z dôvodu, že len teraz bola zakúpená a nestihla jej vedúca knižnice priradiť príslušné číslo.
My sme už dostatočne skúsení databázisti, takže tu teraz nebudem uvádzať dobre známy postup, ako tieto tabuľky vytvoriť.

Typy spojení

Väzby medzi tabuľkami používajú rôzne typy spojení. Väčšina z nich vychádza z normy SQL92, ale každý databázový stroj prináša užívateľovi niektoré viac či menej príjemné zlepšenia.
Typy spojenia v SQL sa rozdeľujú do dvoch hlavných skupín:

- vnútorné spojenia
- vonkajšie spojenia

Vnútorné spojenia

Triviálne spojenie (Trivial Join)

Tento typ spojenia sa tak trochu vymyká z rámca definícií. Ale keďže sa často v literatúre spomína, aj my si ho objasníme. Jedná sa o spojenie jednej tabuľky so sama sebou. Takže sa nejedná o nič iné, ako jednoduchý select danej tabuľky, napr.:
> select * from citatel;

```
mysql> select * from citatel;
+----------+--------+
| meno     | id_lit |
+----------+--------+
| Novak    |      1 |
| Hrbaty   |      2 |
| Janik    |   NULL |
| Bartak   |      4 |
| Horakova |      1 |
+----------+--------+
5 rows in set (1.76 sec)
```

Jednoduché spojenie cez jeden stĺpec
Pozrime sa znovu na tabuľku **CITATEL**. Vidíme, že je v nej uvedené len akési číslo príslušnej literatúry, ktorú si ten-ktorý čitateľ vypožičal. A keďže z tejto tabuľky nevieme, o akú literatúru sa jedná, tak chceme, aby sa vypísal zoznam čitateľov a názov literatúry, ktorú si požičali. Tu už musíme použiť informácie z oboch tabuliek: Použijeme príkaz :

> select * from citatel, literatura where citatel.id_lit = literatura.id_lit;

ktorý hovorí, aby sa vypísali všetky stĺpce z obidvoch tabuliek **CITATEL aj LITERATURA**, ktoré splňujú podmienku, že číslo v stĺpci *id_lit* v tabuľke **CITATEL** sa zhoduje s číslom *id_lit* v tabuľke **LITERATURA**. Ak sa znova pozrieme do vyššie uvedených tabuliek, vidíme, že výpis č. 8-2 je pravdivý, lebo sa obidva stĺpce **id_lit** zhodujú:

```
mysql> select * from citatel, literatura
    -> where citatel.id_lit = literatura.id_lit;
+----------+--------+--------+-------------------+
| meno     | id_lit | id_lit | nazov             |
+----------+--------+--------+-------------------+
| Novak    |      1 |      1 | Amaterske radio   |
| Horakova |      1 |      1 | Amaterske radio   |
| Hrbaty   |      2 |      2 | Teoria relativity |
+----------+--------+--------+-------------------+
3 rows in set (0.06 sec)
```

Čo je to za zvláštny zápis ***citatel.id_lit*** ?
Ak má stĺpec niektorej tabuľky rovnaký názov ako iný stĺpec v inej tabuľke, musíme špecifikovať, z ktorej tabuľky uvažovaný stĺpec vlastne je. Na to sa používa tzv. *"bodkový zápis"* (dobre známy z objektového programovania). Jeho obecný tvar je:

názov_tabuľky.názov_stĺpca

napr.: ***citatel.meno***

V prípade, že by sme chceli spájať tabuľky z rôznych databáz (áno, aj to je niekedy žiadúce), použijeme ešte konkrétnejší zápis:

názov_databáze. názov_tabuľky.názov_stĺpca

napr.: ***kniznica.citatel.meno***

Ak však používame tabuľky len v rámci jednej databázy, vystačíme s menej jednoznačným zápisom.

Vyššie spomenutý výpis príkazu ***SELECT*** môžeme upraviť s použitím slova **JOIN** takto:

>select meno, nazov from citatel JOIN literatura where citatel.id_lit = literatura.id_lit;

Čo táto veta znamená? Voľne by sme ju mohli preložiť asi takto:
Vypíš stĺpce *MENO* a *NAZOV* z tabuľky **CITATEL** spojenej z tabuľkou **LITERATURA**, kde číslo v stĺpci *id_lit* v tabuľke **CITATEL** sa zhoduje s číslom *id_lit* v tabuľke **LITERATURA**. Na výpise č.8-3 je nami požadovaná informácia:

```
mysql> select meno, nazov from citatel JOIN literatura
    -> where citatel.id_lit = literatura.id_lit;
+----------+--------------------+
| meno     | nazov              |
+----------+--------------------+
| Novak    | Amaterske radio    |
| Horakova | Amaterske radio    |
| Hrbaty   | Teoria relativity  |
+----------+--------------------+
3 rows in set (0.00 sec)
```

Z vyššie uvedených príkladov je zrejmé, že konštrukcie s **JOIN** alebo čiarkou (**,**) sú úplne identické.

Určenie podmienky pre spojenie tabuliek
Podmienka

citatel.id_lit = literatura.id_lit

zaisťuje, že vo výsledku bude u každého čitateľa uvedený iba ten riadok tabuľky **LITERATURA**, ktorý obsahuje názov literatúry s rovnakým číslom, ako si vypožičal čitateľ.
Keby sme túto podmienku neuviedli, vykonal by sa **kartézsky súčin** oboch tabuliek. To by znamenalo, že ku každému riadku tabuľky **CITATEL** by sa vyhľadali všetky riadky tabuľky **LITERATURA**. Výsledok takéhoto dopytu bude mať počet riadkov rovný *počtu čitateľov* **krát** *počet literatúry* (5x4=20):

> select * from citatel join literatura;

```
+----------+--------+--------+----------------------+
| Novak    |      1 |   NULL | Sipkova Ruzenka      |
| Hrbaty   |      2 |   NULL | Sipkova Ruzenka      |
| Janik    |   NULL |   NULL | Sipkova Ruzenka      |
| Bartak   |      4 |   NULL | Sipkova Ruzenka      |
| Horakova |      1 |   NULL | Sipkova Ruzenka      |
| Novak    |      1 |      1 | Amaterske radio      |
| Hrbaty   |      2 |      1 | Amaterske radio      |
| Janik    |   NULL |      1 | Amaterske radio      |
| Bartak   |      4 |      1 | Amaterske radio      |
| Horakova |      1 |      1 | Amaterske radio      |
| Novak    |      1 |      2 | Teoria relativity    |
| Hrbaty   |      2 |      2 | Teoria relativity    |
| Janik    |   NULL |      2 | Teoria relativity    |
| Bartak   |      4 |      2 | Teoria relativity    |
| Horakova |      1 |      2 | Teoria relativity    |
| Novak    |      1 |      3 | Cestovny poriadok ZSR |
| Hrbaty   |      2 |      3 | Cestovny poriadok ZSR |
| Janik    |   NULL |      3 | Cestovny poriadok ZSR |
| Bartak   |      4 |      3 | Cestovny poriadok ZSR |
| Horakova |      1 |      3 | Cestovny poriadok ZSR |
+----------+--------+--------+----------------------+
20 rows in set (0.00 sec)
```

Z výpisu č.8-4 vidíme, že nedáva zmysel. Pán Novák si predsa požičal iba Amatérske rádio, ostatnú literatúru nechcel. Nadbytočné riadky môžeme nájsť aj u ostatných čitateľov. Správne sú iba tie riadky, kde sú v oboch číselných stĺpcoch rovnaké hodnoty, teda 6., 10., a 12. riadok výpisu. Preto zadávame pri spojovaní tabuliek podmienku, ktorá omedzí riadky na tie, ktoré chceme dostať do výpisu. Zadávaná podmienka nemusí byť vždy na rovnosť stĺpcov. Často sa používa podmienka príslušnosti do určitého intervalu.

Použitie spojenia tabuliek bez podmienky – kartézskeho súčinu
Existujú prípady, kedy chceme zámerne využiť vlastnosti kartézskeho súčinu - vytvorenie všetkých kombinácií riadkov z oboch tabuliek (metóda každý s každým). Ako už vieme, v tomto prípade nebudeme zadávať žiadnu podmienku pre spojenie oboch tabuliek.

Vonkajšie spojenie

Doteraz sme pracovali s tzv.vnútorným spojením tabuliek, ktorému sa anglicky hovorí **INNER JOIN**. Do výsledného výpisu boli zahrnuté iba tie riadky z oboch tabuliek, pre ktoré bola nájdená odpovedajúca hodnota v druhej tabuľke. Pozrime sa opäť na výpis č. 8-3. Vidíme, že do výsledku neboli zahrnutí tí čitatelia, ktorí nemajú zadané číslo literatúry (Janík), alebo ich vypožičaná literatúra neexistuje (Barták). Naviac vo výsledku nie je ani literatúra, ktorú si nikto nepožičal (Cestovný poriadok ŽSR), alebo ešte nema pridelené číslo (Šípková Ruženka).
(**POZOR!** Podmienka ***citatel.id_lit = literatura.id_lit*** nie je splnená ani v prípade, kedy ***citatel.id_lit = NULL*** (Janik) a ***literatura.id_lit =NULL*** (Sipkova Ruzenka) v 3.riadku výpisu č.8-4. Výsledkom porovnania je totiž zase NULL, nie hodnota TRUE (áno) potrebná pre potvrdenie podmienky rovnosti.)

Ľavé vonkajšie spojenie
Existuje spôsob spojenia, ktorý umožňuje zaradiť do výsledku aj tie riadky, pre ktoré nebola nájdená odpovedajúca hodnota v druhej tabuľke. Takto môžeme vypísať zoznam kníh a im odpovedajúcich čitateľov, v ktorom bude uvedená aj literatúra, ktorú si nik nepožičal. Alebo naopak, vypíšeme zoznam všetkých čitateľov, ktorí si nepožičali žiadnu literatúru. Takémuto spojeniu sa hovorí ***vonkajšie spojenie***.
Tieto vonkajšie spojenia môžu byť dvojakého druhu - **ľavé** (LEFT) a **pravé** (RIGHT). MySQL server podporuje iba **ľavé** vonkajšie spojenie. (A my si ukážeme, ako dosiahnúť pravé spojenie).
Ľavé vonkajšie spojenie vytvoríme konštrukciou **LEFT JOIN**. Použitím tohto spojenia dosiahneme vo výslednom výpise zahrnutie všetkých riadkov z ľavej (teda z prvej) tabuľky. Ak nebol nájdený odpovedajúci riadok v pravej tabuľke, budú vo výsledku hodnoty NULL vo všetkých stĺpcoch použitých z druhej tabuľky. Použitá konštrukcia je zrejmá z výpisu č.8-5:

```
mysql> select meno, nazov from citatel LEFT JOIN literatura
    -> ON citatel.id_lit = literatura.id_lit;
+----------+--------------------+
| meno     | nazov              |
+----------+--------------------+
| Novak    | Amaterske radio    |
| Hrbaty   | Teoria relativity  |
| Janik    | NULL               |
| Bartak   | NULL               |
| Horakova | Amaterske radio    |
+----------+--------------------+
5 rows in set (0.16 sec)
```

Vidíme, že páni Janík a Bartak majú v pravom stĺpci hodnotu NULL. Janík si nič nepožičal a Barták má číslo pôžičky, ktorá neexistuje.
Pred chvíľou sme si povedali, že ak máme rovnaké názvy stĺpcov v jednotlivých tabuľkách, pre jednoznačnosť musíme použiť bodkový zápis. V prípade ľavého spojenia si však zápis môžeme zjednodušiť konštrukciou **USING (*meno_porovnávaného_stĺpca*)** tak, ako je to na výpise č.8-6:

```
mysql> select meno, nazov from citatel LEFT JOIN literatura
    -> USING (id_lit);
+----------+--------------------+
| meno     | nazov              |
+----------+--------------------+
| Novak    | Amaterske radio    |
| Hrbaty   | Teoria relativity  |
| Janik    | NULL               |
| Bartak   | NULL               |
| Horakova | Amaterske radio    |
+----------+--------------------+
5 rows in set (0.06 sec)
```

LEFT OUTER JOIN
Je ekvivalent príkazu **LEFT JOIN**. Používa sa pre kompitibilitu s drivermi ODBC.

Pravé vonkajšie spojenie
Použitím tohto spojenia dosiahneme vo výslednom výpise zahrnutie všetkých riadkov z pravej (teda z druhej) tabuľky. Ak nebol nájdený odpovedajúci riadok v ľavej tabuľke, budú vo výsledku hodnoty NULL vo všetkých stĺpcoch použitých z prvej tabuľky. Ako vieme, MySQL nepodporuje pravé spojenie. My ho však vieme vytvoriť z ľavého spojenia jednoduchým prehodením poradia spájaných tabuliek okolo príkazu LEFT JOIN. Použitá konštrukcia je zrejmá z výpisu č.8-7:

```
mysql> select meno, nazov from literatura LEFT JOIN citatel
    -> USING (id_lit);
+----------+-----------------------+
| meno     | nazov                 |
+----------+-----------------------+
| NULL     | Sipkova Ruzenka       |
| Novak    | Amaterske radio       |
| Horakova | Amaterske radio       |
| Hrbaty   | Teoria relativity     |
| NULL     | Cestovny poriadok ZSR |
+----------+-----------------------+
5 rows in set (0.00 sec)
```

Tu vidíme, že Šípková Ruženka a Cestovný poriadok ŽSR obsahujú v ľavom stĺpci, kde sa nachádza meno čitateľa, hodnotu NULL, lebo tá prvá ešte nemá pridelené číslo a cestovný poriadok si nik nepožičal.

V praxi sa veľmi často používa práve ľavé *(LEFT JOIN*) spojenie.

Technické detaily spojovania tabuliek

Spojovanie tabuliek je veľmi časovo náročná činnosť. Preto je veľmi vhodné zamyslieť sa nad spôsobom, ktorým sú tabuľky spojované a následne sa pokúsiť vhodným zadaním príkazu SELECT zrýchliť jeho spracovanie. Pri spojovaní tabuliek sa jedná obecne o kartézsky súčin dvoch relácií. Výsledný počet záznamov bude rásť exponenciálne s počtom riadkov vo vstupujúcich tabuľkách. Naším základným cielom by preto malo byť čo najviac omedziť počet riadkov v tabuľkách ešte pred vlastným spojovaním. Napr. ak chceme zistiť informácie o knižniciach v Trenčianskom kraji, obmedzíme najprv tabuľku knižníc iba na knižnice v tomto regióne a až potom výsledok budeme spojovať s tabuľkami kníh, autorov a iné. Ďalej je veľmi dôležité, aby sme vždy použili obmedzenia pre spojované stĺpce (v JOIN alebo WHERE), a tým čo najviac znížili počet riadkov vo výslednom výpise. Našťastie, dnešné databázové stroje majú zabudované veľmi rýchle spojovacie mechanizmy, ktoré mnohé optimalizácie vykonávajú automaticky bez nášho zásahu.

Naše spojenie
Vráťme sa však k našej databáze KNIZNICA a tabuľkám KNIHA a ZANER. Vieme, že tabuľka KNIHA obsahuje stĺpec CIS_ODD, ktorým sa odvoláva na podobný stĺpec v tabuľke ZANER.
Použijeme už známu konštrukciu LEFT JOIN USING.

```
mysql> select id, nazov, autor, tematika
    -> from kniha left join zaner using (cis_odd);
+----+------------------------------+-----------------------+--------------+
| id | nazov                        | autor                 | tematika     |
+----+------------------------------+-----------------------+--------------+
|  1 | Angelika a kral              | Golon, Anne a Serge   | roman        |
|  2 | KGB                          | Gordijevsky, Oleg     | lit. faktu   |
|  3 | Bratia Ricovci               | Simenon, Georges      | krimi        |
|  4 | Vtaky v trni                 | McCulloughova, Collen | roman        |
|  5 | Linux - prakticky pruvodce   | Sobell, Mark G.       | odborna lit. |
|  6 | Naucte se programovat v Delphi | Binzinger, Thomas   | odborna lit. |
|  7 | Pouzivame linux              | Welsh, M., Kaufman, L. | odborna lit. |
|  8 | Z polovnickej kapsy          | Moric, Rudo           | detska lit.  |
|  9 | Plebejska kosela             | Mihalik, Vojtech      | poezia       |
| 10 | Europou bez penazi           | Hlubucek, Petr, Ing.  | cestopis     |
+----+------------------------------+-----------------------+--------------+
10 rows in set (0.22 sec)
```

Na výpise č.8-8 vidíme už podstatne prijateľnejší výpis, aký sme používali v minulých častiach seriálu.

Na dnes by stačilo, a nabudúce sa začneme zaoberať jednou z najdôležitejších činností SQL serveru - jeho administráciou.

Malé veľké databázy – 9.časť

Dnes si spolu prejdeme skutočne najdôležitejšiu vec v každom SQL serveri, teda MySQL nevynímajúc. Sú to prístupové práva. Práve tie robia SQL server tým spoľahlivým systémom. Bez nich by žiadny SQL systém ani nemal zmysel. Pevne verím, že to dnes spolu zvládneme.

Poznámka k práci s touto lekciou:
Nedajte sa odradiť, ak niečomu neporozumiete hneď. Preberte si jednotlivé odstavce postupne a pomaly a snažte si uvedomiť základné pojmy a významy. Pozor na to, kedy hovoríme o tabuľke **USER**, **DB** a **HOST** a kedy o položke ***user, db*** a ***host*** v týchto tabuľkách. (Môj veľký obdiv Montymu – autorovi MySQL - je trošičku zakalený preto, že – podľa môjho názoru – nevhodne zvolil názvy jednotlivých tabuliek a položiek v nich. No uznajte, **mysql** je názov SQL systém, ale aj databáze grant tabuliek a zároveň názov klienta, **user** je tabuľka aj položka v nej a pod.). Pre názornosť budem názvy tabuliek uvádzať **VEĽKÝMI** a názvy položiek v jednotlivých tabuľkách ***malými*** písmenami. Pri čítaní kapitol sa stále pozerajte do grant tabuliek, aby ste si uvedomili, o ktorej položke alebo privilégiu je reč. Na kúsok papiera si značte ceruzkou väzby medzi jednotlivými tabuľkami a položkami. Aj ja som mal pri štúdiu okolo seba samé tabuľky a papiere s poznámkami a priznávam, dosť dlho som v tom „ležal“. Vyskúšajte si vzorové príklady, nastavte práva, reloadnite tabuľky a skúšajte sa potom konektovať na server. Dopredu si predstavte, aké výsledky by ste mali dostať a potom ich skontrolujte so získanými. Ak súhlasia, tak ste pokročili, ak nie, vráťte sa na začiatok. Skúšať, skúšať, skúšať. (A či učiť sa, učiť sa, učiť sa ?!?)

Trocha teórie

Prístupové práva
Už sme si raz spomínali, že nie je žiadúce, aby v databázovom systéme mohol robiť každý všetko. Pozrime sa na to opäť z pohľadu našej imaginárnej knižnice. Vieme, že napr. vedúca knižnice potrebuje do centrálnej databáze pridávať novo zakúpené knihy alebo rušiť knihy už veľmi opotrebované. Z titulu funkcie to môže urobiť ona a len ona. Zato ale referentka knižnice, ktorá vedie výpožičku kníh, potrebuje pridávať mená nových zákazníkov, zaznamenávať, aké knihy má kto zapožičané a samozrejme aj po vrátení kníh ich odpisovať z evidencie. Naproti tomu obyčajný čitateľ môže len prezerať, aké knihy sú v knižnici k dostaniu a ktoré sú práve voľné a kde sa nachádzajú. No, a aby sme to mali kompletné, pán riaditeľ – ten môže samozrejme všetko. Teda skoro.
Aby sme mohli v našom systéme zadefinovať tieto pravidlá, musíme ich obecne definovať v akýchsi **tabuľkách prístupových práv.**

Tabuľky prístupových práv
Aj MySQL má také tabuľky prístupových práv. Hovorí sa im aj **grant tabuľky – grant tables**. Nachádzajú sa v databáze s názvom **mysql**. Do tejto databáze sa prepneme v monitore mysql príkazom ***use mysql*** (použi mysql databázu), a server hláškou ***Database changed*** (Databáza zmenená) potvrdí vykonanú zmenu.
Zadáme príkaz ***show tables*** a uvidíme 5 tabuliek s týmito názvami:
- USER
- DB
- HOST
- TABLES_Priv
- COLUMNS_Priv

Prvé tri tabuľky **USER, DB** a **HOST** sú základné, ostatné dve sú doplnkové tabuľky práv. Dnes sa budeme zaoberať iba základnou skupinou, ktorá vo väčšine projektov plne postačuje. Doplnkové tabuľky si preberieme, keď zvládneme základnú tématiku.
Keby sme na každú tabuľku použili nám už známy príkaz ***describe table***, napr. ***describe USER***, dostaneme popis jednotlivých tabuliek. Stručný prehľad položiek v jednotlivých databázach je v tab.č.9-1:

Tab.č.9 - 1: **Štruktúra tabuliek USER, DB a HOST**

Tabuľka	USER	DB	HOST
Autentifikácia	Host	Host	Host
	User	Db	Db

	Password	User	
Databázové a	Select_Priv	Select_Priv	Select_Priv
tabuľkové privilégiá	Insert_Priv	Insert_Priv	Insert_Priv
	Update_Priv	Update_Priv	Update_Priv
	Delete_Priv	Delete_Priv	Delete_Priv
	Create_Priv	Create_Priv	Create_Priv
	Drop_Priv	Drop_Priv	Drop_Priv
Administrátorské	Reload_Priv		
privilégiá	Shutdown_Priv		
	Process_Priv		
	File_Priv		
	Grant_Priv	Grant_Priv	Grant_Priv
	References_Priv	References_Priv	References_Priv
	Index_Priv	Index_Priv	Index_Priv
	Alter_Priv	Alter_Priv	Alter_Priv

Musím upozorniť, že každá verzia MySQL môže mať inú štruktúru jednotlivých tabuliek. Preto je **NUTNÉ** použiť príkaz ***describe***, aby sme zistili požadované položky. Ja používam verziu 3.23-27-beta, a preto tieto príkazy budú šité na túto verziu. V iných verziách rádu 3.22 sa počty parametrov môžu líšiť (a spravidla sa aj líšia. A aby to nebolo ešte všetko, tak sa verziu od verzie mení aj poradie parametrov v časti privilégií.).

V priebehu dnešnej kapitoly a samozrejme vždy pri práci s právami sa budeme do týchto tabuliek pozerať veľmi, veľmi často. Preto je vhodné si ich uložiť niekde poruke.

Autentifikácia
Už aj z iných bezpečných systémov vieme, že sa každý užívateľ pri vstupe do systému musí autentifikovať. To sa vykonáva spravidla zadaním mena a hesla. Na základe tejto autentifikácie sa každému prihlásenému užívateľovi pridelia príslušné prístupové práva. Tu si musíme uvedomiť jednu vec. Meno a heslo užívateľa v MySQL nie je totožné s užívateľom v danom operačnom systéme, pod ktorým SQL server beží. Dokonca tento užívateľ vôbec nemusí byť v operačnom systéme vôbec definovaný. Táto výhoda sa uplatní práve v tých systémoch, kde sa za každého licencovaného užívateľa operačného systému tvrdo platí, napr. SCO Unix alebo Windows NT. To znamená, že aj keď máme licenciu len na 15 užívateľov SCO, v MySQL môžeme mať nadefinované stovky užívateľov databázovej aplikácie.
Mená a heslá užívateľov sú uložené v tabuľke **USER** v položke ***user*** a ***password***. Heslá sa neukladajú v otvorenom tvare, ale kryptované funkciou *password()*. Pri autentifikácii sa zadané heslo zakryptuje a porovná s heslom v tabuľke.
Každú z grant tabuliek si môžeme rozdeliť na tri časti:
Prvá časť je autentifikačná, kde sa definuje kto (user), pod akým heslom (password), zkadiaľ (host) a kde (db) môže prístúpiť v SQL serveri.
Druhú časť tvoria privilégia k databázam a tabuľkám, ktoré určujú, čo môže pripojený a už autentifikovaný užívateľ vykonávať v danej databáze a tabuľkách.
Tretiu časť tvoria administrátorské privilégiá. Tieto určujú, čo môže pripojený užívateľ konať so samotným SQL serverom.
Význam a popis jednotlivých privilégií je v tab.č.9-2:

Tab.č.9 - 2 : **Privilégiá**

Privilégium:	**Umožňuje:**
Select_Priv	prezeranie tabuľky
Insert_Priv	vkladanie do tabuľky
Update_Priv	zmeny v tabuľke
Delete_Priv	vyprázdňovanie - mazenie v tabuľke

Create_Priv	vytvorenie databáz a tabuliek
Drop_Priv	zmazanie databáz a tabuliek
Reload_Priv	znovunačítanie grant tabuliek
Shutdown_Priv	ukončenie činnosti servera
Process_Priv	výpis procesov
File_Priv	čítanie a zápis do súboru
Grant_Priv	schopnosť odovzdať svoje práva iným
References_Priv	nevyužité
Index_Priv	tvoriť alebo mazať indexy
Alter_Priv	meniť štruktúru tabuliek

Obsah položiek
V autentifikačnej časti každej tabuľky sa môžu nachádzať rôzne znaky. V položkách ***user, host*** a ***db*** sa nachádza spravidla text, vyjadrujúci konkrétnu hodnotu, napr: *user = 'balu', db ='kniznica' a host='server.niekde.sk'*, čo značí, že uživateľ s menom *balu* zo servera *server.niekde.sk* sa môže pripojiť do databáze *knižnica*.
Heslo je tiež typu char, avšak je kryptované funkciou *password* a pri bežnom selekte sa javí ako skupina hexadecimálnych čísiel.
Okrem bežného textového reťazca sa v týchto položkách môžu nachádzať ešte špeciálne znaky, čo si povieme neskôr.
Pozor! Hodnoty v položkách ***user, password, db*** sú case sensitive, teda rozlišujú veľké a malé písmená. Hodnoty v položke ***host*** sú case insensitive, teda nerozlišujú veľkosť písmen.
Toto býva častou chybou, kedy si užívateľ myslí, že zadal všetko správne, ale aj tak sa nepripojil. A ono to bolo práve v malých a veľkých písmenách.
V privilegačnej časti sa používajú iba dva znaky – **"Y"** a **"N"** ako *yes* a *no*, teda či má konkrétny užívateľ dané privilégium alebo nie.

Prístup na server
Prístup na SQL server a k jednotlivým databázam prebieha v dvoch krokoch:
- verifikácia spojenia
- verifikácia požiadaviek

Verifikácia spojenia
Verifikácia – overenie – spojenia vychádza z kontroly identity užívateľa. Identita užívateľa je definovaná v týchto informáciách v tabuľke **USER**:
- názov počítača, z ktorého užívateľ pristupuje – položka ***host***
- meno a heslo užívateľa – položky ***user*** a ***password***

Pri pokuse o spojenie server najprv vykoná overenie totožnosti v týchto položkách. Ak súhlasia, spojenie sa vykoná, ak nie, spojenie sa odoprie.
Jednotlivé položky tabuľky **USER** môžu nadobúdať tieto hodnoty:
host položka:
- Host hodnota môže byť meno alebo IP adresa počítača - klienta, z ktorého sa uskutočňuje spojenie. Tiež to môže byť reťazec ***"localhost"***, ktorý indikuje lokálny – miestny – počítač. Localhost je taký host, kde na danom operačnom systéme pracuje SQL server aj klient zároveň.
- môžeme použiť aj tzv. "žolíky" (a to znak **%** - percento a **_** podtržítko) a prázdny znak **""**.
  V tomto prípade hodnota **"%"** znamená ľubovoľný počíťač - klient. Prázdny znak, teda prázdne políčko stĺpca host, je ekvivalent **„%"**. Uvedomme si, čo toto znamená: Každý klient - počítač môže vykonať spojenie na náš server!

user položka:
- žolíky nemôžeme použiť v položke ***user***, pretože **„%"** značí užívateľa s menom **„%"**
- môžeme použiť prázdny znak, čo indikuje *„ľubovoľný (každý) užívateľ"*.

Hovorí sa mu aj ***anonymný užívateľ*** (anonymous). To znamená, že môžeme zadať ľubovoľné meno. V praxi nie je vhodné definovať anonymného užívateľa, a ak áno, tak len s veľmi obmedzenými právami. Súvisí to s bezpečnostnou politikou, ktorú si teraz nebudeme preberať.

password položka:

- heslo môže byť kryptované alebo prázdne. Prázdne heslo neznamená, že je možné zadať ľubovoľné heslo, ale značí, že užívateľ sa konektuje bez zadania hesla.

Tabuľka č.9-3 obsahuje príklady niektorých kombinácií hodnôt ***host*** a ***user*** tabuľky **USER**:

Tab.č.9 -3: **Príklady položiek host a user**

Položka host	user	Popis
"doma.niekde.sk"	"mior"	užívateľ mior z počítača doma.niekde.sk
"doma.niekde.sk"	""	ľubovoľný užívateľ z počítača doma.niekde.sk
"%"	"mior"	užívateľ mior z ľubovoľného počítača
"%"	""	ľubovoľný užívateľ z ľubovoľného počítača
"%.niekde.sk"	"mior"	užívateľ mior z ľubovoľného počítača v doméne niekde.sk
"192.168.10.153"	"mior"	užívateľ mior z počítača s IP adresou 192.168.10.153
"192.168.10.%"	"mior"	užívateľ mior z ľubovoľného počítača v podsieti 192.168.10.xx

Ako to funguje?

MySQL server si prečíta tabuľky prístupových práv vtedy, keď štartuje. Preto je potrebné pri každej zmene týchto tabuliek vykonať ich nové načítanie. Nemusíme preto server zastavovať a znova štartovať, stačí, ak vykonáme tzv. reload grant tabuliek. To môžeme vykonať priamo z mysql monitora príkazom:

mysql> flush privileges;

alebo v operačnom systéme zadaním príkazu:

mysqladmin reload

Pozor! Na vykonanie takejto operácie musíme mať dovolené privilégium ***reload_priv***. Zatiaľ predpokladajme, že sme užívateľ *root*, ktorému je všetko dovolené.

Keď sa pozrieme do tab.č. 9 – 4, vidíme, že pán riaditeľ sa môže pripájať k MySQL serveru z troch pozícií – buď zo svojho počítača v kancelárii alebo z lokálnej konzoly servera alebo z ľubovoľného počítača v sieti. Ale ktoré práva teda bude mať?

Tab.č.9 - 4: **Prístup užívateľa *riaditel***

host	user	Select_Priv	Insert_Priv	Update_Priv	Delete_Priv	Create_Priv	Drop_Priv	Reload_Priv	Shutdown_Priv	Process_Priv	File_Priv	Grant_Priv	References_Priv	Index_Priv	Alter_Priv
"%"	" riaditel "	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N
"%"	""	Y	N	N	N	N	N	N	N	N	N	N	N	N	N
"pracovna"	"riaditel"	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
"localhost"	"riaditel"	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

SQL server pri štarte alebo reloade (znovunačítaní tabuliek práv) nielen že tabuľky načíta, ale si ich aj vnútorne striedi od najkonkrétnejšieho k najvšeobecnejšiemu zápisu podľa tohto princípu:

- ako prvé usporiada záznamy, kde ***host*** neobsahuje žolíky
- nasledujú záznamy v položke ***host*** , kde súv texte žolíky, napr. *%.niekde.sk*
- na konci sú záznamy s "%" alebo prázdne záznamy. (Už vieme, že prázdny záznam je identický s "%").
- ak sú záznamy v ***host*** identické, položka ***user*** s konkrétnym menom má prednosť pred prázdnou položkou

Takže tabuľku č.9-4 server usporiada tak, ako je to v tabuľke č.9-5:

Tab.č.9 - 5: **Prístup užívateľa *riaditel* potriedení**

host	user	Select_Priv	Insert_Priv	Update_Priv	Delete_Priv	Create_Priv	Drop_Priv	Reload_Priv	Shutdown_Priv	Process_Priv	File_Priv	Grant_Priv	References_Priv	Index_Priv	Alter_Priv
"localhost"	"riaditel"	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
"pracovna"	"riaditel"	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
"%"	"riaditel"	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N
"%"	""	Y	N	N	N	N	N	N	N	N	N	N	N	N	N

Server potom číta od vrcholu tabuľky, a keď narazí na identitu, ktorá patrí danému užívateľovi, priradí mu aj zodpovedajúce práva, definované pri tejto identite. Ostatné identity toho istého užívateľa, ktoré sú v tabuľke definované nižšie, už ignoruje. Takže, pohľadom do tab.č.9-5 vidíme, že pán riaditeľ, ak bude pracovať na lokálnej konzole – potažmo klávesnici počítača, kde beží MySQL server aj klient, bude mať všetky práva (1.riadok tab.č.9-5), ale ak bude pracovať za svojím počítačom v kancelárii, budú už jeho práva čiastočne obmedzené (2.riadok tab.č.9-5). Keď si však sadne za počítač v čitárni knižnice, bude si môcť iba jednotlivé tabuľky prezerať (Select_priv = "Y") a vkladať do nich príslušné záznamy (Insert_priv = "Y"), ale veci spojené s administráciou tabuliek už nemôže vykonať (3.riadok tab.č.9-5). Ak si za tento počítač sadne ľubovoľný čitateľ knižnice, zadá svoje meno, ktoré nie je definované v tabuľke alebo ho vôbec nevyplní (teda User = ""), je považovaný za anonymného užívateľa a bude môcť záznamy iba prezerať (4.riadok tab.č.9-5).

Verifikácia požiadaviek

Ak je už raz spojenie etablované, pristúpi server k verifikácii požiadaviek. Teda overí, či na tú operáciu, ktorú sa užívateľ zrovna chystá vykonať, má privilégium. Napr., ak chce užívateľ selektovať nejakú tabuľku, server najprv skontroluje, či má tento užívateľ nastavené privilégium ***Select_priv***. Toto privilégium môže byť nastavené v niektorej z tabuliek **USER, DB** alebo **HOST**.

Pri verifikácii požiadaviek server prezerá nastavené privilégia na tomto princípe:

Veľmi záleží na poradí tabuliek. SQL server filtruje tabuľky v tomto poradí:

- **USER** tabuľku
- **DB** tabuľku
- **HOST** tabuľku

Teda ako prvé sa server pozrie na privilégiá do grant tabuľky **USER**.

USER tabuľka prisudzuje privilégiá v **globálnom** merítku na celý SQL server. Napr. ak má určitý užívateľ nastavené privilégium ***Delete_priv*** (teda ***Delete_priv* =*"Y"***) v tejto grant tabuľke**,** môže mazať záznamy vo všetkých databázach na MySQL serveri!

Inými slovami – privilégiá v **USER** tabuľke sú ***superužívateľské*** privilégiá. Preto tieto privilégiá prideľujme len superužívateľom, ako je serverový alebo databázový administrátor. Takému superužívateľovi sa v MySQL hovorí **root**. Samozrejme, ten môže tieto práva preniesť aj na iného dôveryhodného užívateľa, ktorý ho bude v neprítomnosti zastupovať. Avšak nemusia to byť úplné práva, to však súvisí s už spomínanou bezpečnostnou

politikou. (Tak teda len na okraj: každý dobrý projekt musí mať vypracovanú bezpečnostnú politiku. Tá definuje, kto, čo, kedy, ako, skadiaľ, prečo ..., a samozrejme aj spôsob archivácie dát, obnovu dát po krachu servera, ako aj ochranu proti hackerom a iné. Čert NIKDY nespí! Ale o tom možno inokedy.)
Ostatným užívateľom necháme v tabuľke **USER** **nenastavené** globálne privilégiá (teda "**N**") a na definovanie ich lokálnych prístupových práv k jednotlivým databázam a tabuľkám použijeme tabuľky **DB** a **HOST**.

Tabuľky DB a HOST
Tabuľky **DB** a **HOST** poskytujú privilégiá špecifické ku konkrétnym databázam. Hodnoty v jednotlivých autentifikačných poliach môžu byť špecifikované takto:
- žolíky ("%" a "_") môžu byť definované v ***host*** a ***db*** poliach obidvoch tabuliek
- "%" v položke ***host*** v tabuľke **DB** znamená „ľubovoľný klientský počítač"
- prázdny znak "" v položke ***host*** v tabuľke **DB**, (teda je prázdna), znamená : Pozri sa do tabuľky **HOST** pre ďalšie informácie.
- "%" alebo prázdny znak v položke ***host*** v tabuľke **HOST** znamená „ľubovoľný klientský počítač"
- "%" alebo prázdny znak v položke ***db*** v obidvoch tabuľkách **DB** a **HOST** znamená „ľubovoľná databáza".
- prázdny znak v položke ***user*** v obidvoch tabuľkách značí anonymného užívateľa

Obidve tabuľky **DB** aj **HOST** sú čítané a triedené vtedy, keď server štartuje, teda v ten istý čas, keď **USER** tabuľka.
DB tabuľka je triedená podľa ***host, db*** a ***user*** položiek, **HOST** tabuľka je triedená podľa ***host*** a ***db*** položiek. Tak ako **USER** tabuľka, aj tieto sú triedené princípom od najkonkrétnejšieho po najobecnejší záznam. Samozrejme, platí prvý nájdený a zodpovedajúci triedený záznam.
Pre administratívne požiadavky (*shutdown, reload* a pod.) SQL server skontroluje len tabuľku **USER**, pretože sa inde tieto privilégiá nenachádzajú. Napr. ak užívateľ chce vykonať príkaz **mysqladmin shutdown**, ale v **USER** tabuľke nemá nastavené ***Shutdown_priv***, príkaz sa odoprie.

Ak sa požadujú operácie s databázou alebo tabuľkou (*select, insert, update* a pod.), server najprv pozrie do tabuľky **USER** na globálne superužívateľské práva. Ak privilégium v **USER** tabuľke umožňuje požadovanú operáciu, tak sa prístup zabezpečí.
Ak sú privilégia v **USER** tabuľke nedostatočné, server sa pozrie do tabuliek **DB** a prípadne **HOST**, kde sa podľa nastavených lokálnych práv rozhodne o umožnení alebo zamietnutí prístupu.
Najprv sa pozrie do tabuľky **DB** a rozhoduje sa takto:
- pozrie na položky ***host, db*** a ***user***. Položka ***db*** obsahuje názov databáze, kde chce užívateľ pristúpiť. Ak nie je žiadny zodpovedajúci záznam v položke ***host*** a ***user*** **alebo** ***db***, prístup sa odmietne.
- ak existuje zodpovedajúci záznam v položke ***user*** (teda meno užívateľa platí) a ***host*** nie je prázdny, ale konkrétny a správny, tak sa priznajú stanovené lokálne privilégiá.
- ak existuje zodpovedajúci záznam v položke ***user*** a položka ***host*** je prázdna, server sa pozrie do tabuľky **HOST** a prehľadá tam položky ***host*** a ***db*** v tejto tabuľke. Ak v nich nie sú zodpovedajúce záznamy, prístup sa odmietne. Ak sa tu nachádzajú zodpovedajúce záznamy, privilégiá sa počítajú v intersekcii s privilégiami v **DB** a **HOST** tabuľke. Teda obidve privilégiá musia byť „**Y**".

Zložité, však. A na čo je to dobré? My totiž môžeme priradiť všeobecné lokálne (pozor! nie administrátorské!!!) privilégiá v **DB** tabuľke a potom ich obmedzovať pre jednotlivých klientov v tabuľke **HOST**.

Vyjadrené logicky, užívateľové privilégiá sú kalkulované takto:

privilégiá v **USER** tabuľke *ALEBO* privilégiá v **DB** tabuľke *A ZÁROVEŇ* privilégiá v **HOST** tabuľke,

alebo matematicky

USER *OR* (**DB** *AND* **HOST**).

Po zmene nastavení nesmieme zabudnúť tabuľky reloadnúť.

A trocha príkladov

Nastal čas, aby sme si vyššie uvedenú teóriu objasnili v praxi. A ako sa vlastne s grant tabuľkami narába? No predsa presne tak, ako s ľubovoľnou tabuľkou , teda použijeme nám dobre známe príkazy *select, insert, update, delete* a iné. Tak ideme na to!

Začneme tým, že si vyprázdnime všetky grant tabuľky v databáze **mysql**. Užívateľ **root** je od inštalácie MySQL nastavený ako *superuser*, teda má všetky globálne práva na všetky databázy a ich tabuľky. Ak sme si v prvých lekciách zmenili jeho heslo, spustíme mysql monitor s parametrom mysql, aby sme sa vnorili do databáze mysql takto:
mysql –u root –p*heslo_pre_roota* mysql

alebo ak nemáme nadefinované heslo pre roota, tak iba:
mysql –u root mysql

Sme pripojení na server a sme nastavení v databáze mysql.

Teraz vyprázdnime tabuľku **USER** príkazom:
mysql> delete from USER;

Potom vložíme prvého užívateľa **root** s príslušným heslom a všetkými globálnymi právami:

mysql> insert into USER values('localhost','root',password('heslo'),
'Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y');

Za ***heslo*** si doplníme každý to svoje heslo!
Skontrolujeme pomocou:

mysql> select * from USER;

či sú práva nastavené tak, ako sme chceli.

Podobne vymažeme tabuľky **DB**, **HOST** a aj zatiaľ nepreberané tabuľky **TABLES_Priv** a **COLUMNS_Priv** (to aby nás nemýlili).

Keď sme tak učinili, zadáme:

mysql> flush privileges;

čím znova načítame grant tabuľky.

Ukončíme mysql monitor príkazom *exit* alebo *quit* a pokúsime sa znova konektovať na server. Skúsime to najprv bez mena alebo s použitím iného, ľubovoľného mena, napr. *monty*:

mysql (bez parametrov)
alebo
mysql –u monty

Server sa pozrie do tabuľky **USER**, a keďže tam nie je definovaný *monty* ani iný *anonymous*, prístup odoprie – pozri obr.č.9-1:

```
C:\mysql\bin>mysql -u monty
ERROR 1045: Access denied for user: 'monty@localhost' (Using password: NO)

C:\mysql\bin>_
```

Teraz zadáme správne parametre, čiže:

mysql –u root –p*stanovené_heslo*

a sme tam (obr.č.9-2):

```
C:\mysql\bin>mysql -u root -pheslo
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 3.23.27-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> _
```

Nastavíme sa do databáze **mysql** použitím

mysql> use mysql;

(Mohli sme to urobiť aj ako parameter príkazu mysql priamo na príkazovom riadku operačného systému, teda **mysql –u root –pstanovené_heslo mysql**).

Vložíme nových užívateľov takto:

mysql> insert into USER values('localhost','riaditel',password('direktor'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

Keďže vieme, že v privilégiách sú defaultne nastavené hodnoty "N", nemusíme zadávať všetkých štrnásť "N", ale stačí, ak zápis zjednodušíme

mysql> insert into USER (host, user, password) values('localhost','riaditel',password('direktor');

[
(Kto má možnosť práce na sieti, môže si skúsiť pripojenie z rôznych počítačov, len si zmení meno počítača v položke ***host*** podľa potreby:

mysql> insert into USER values('pracovna','riaditel',password('direktor'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into USER values('citaren','riaditel',password('direktor'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

Takto sme definovali pána riaditeľa, ktorý môže pristúpiť k SQL serveru z troch rôznych počítačov, tak ako sme si to prezentovali v teoretickej časti. Všimnime si, že aj keď to je pán riaditeľ, neudelili sme mu globálne, teda superuserské práva. Avšak jeho najvyšie postavenie v knižnici budeme definovať práve v tabuľke **DB**.

Väčšina ale nemá možnosť práce na sieti, teda my budeme pokračovať skúšaním na „lokále".
]

Vložíme ďalších troch užívateľov, *veduca*, *referent* a *anonymous*:

mysql> insert into USER values('localhost','veduca',password('sefka'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into USER values('localhost','referent',password('pracant'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into USER values('localhost','','',
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

Ani týmto dámam a knihomoľovi - anonymousovi sme nepridelili superuserské práva.

Vykonáme ***flush privileges*** a ukončíme mysql monitor.
Znova vyskúšame spojenie k SQL serveru, trebárs ako pán riaditeľ takto:

mysql –u riaditel –pdirektor

Čo sa stane? Server skontroluje, či v tabuľke **USER** existuje užívateľ ***riaditel*** s heslom ***direktor***. Keďže áno, spojenie uskutoční. Ale: Skúsme teraz meniť nejakú databázu, napr.:

mysql> use mysql;

a vidíme, že server odmietol túto požiadavku, tak ako je to vidieť na výpise č.9-3:

```
C:\mysql\bin>mysql -u riaditel -pdirektor
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17 to server version: 3.23.27-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> use mysql;
ERROR 1044: Access denied for user: 'riaditel@localhost' to database 'mysql'
mysql> use kniznica;
ERROR 1044: Access denied for user: 'riaditel@localhost' to database 'kniznica'
mysql>
```

A je to správne, pretože pán riaditeľ nemá žiadne globálne ani lokálne práva, teda zatiaľ. Vyskúšajme zmeniť aj iné databázy, napr. **kniznica**. Výsledok je rovnaký.
Podobné výsledky dostaneme, ak sa bude konektovať užívateľ *veduca* , *referent* či *citatel*.

No, globálne práva sme zakázali, ale teraz musíme nastaviť lokálne práva.
Ako sme si v teórii povedali, lokálne práva nastavujeme v datábáze **DB** a **HOST**.

Znovu sa pripojíme k databázi **mysql** ako *root* príkazom:

mysql –u root –p*stanovené_heslo*

a príkazom :

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'riaditel', 'Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y','Y','Y','Y');

definujeme, že pán riaditeľ môže z lokálneho počítača pristupovať k databazi **kniznica** a má plné (lokálne) databázové privilégiá.
Znova "flušneme" privilégiá a ukončíme mysql monitor. Opätovne sa pripojíme k serveru, teraz ako užívateľ *riaditel*. Ide to. Skúsime použiť databázu, napr. **mysql**. Server odmietne a my už vieme, prečo.
Ale teraz použijeme databázu **kniznica**.
Čo sa stane? Server skontroluje identitu v tabuľke **USER**. Zistí, že pripojovaný užívateľ je definovaný, meno a heslo je platné. Keďže nemá stanovené žiadne globálne privilégiá, nazrie do databáze **DB**, či tam nie je užívateľ s menom *riaditel* definovaný pre databázu **kniznica**. Zistí, že áno a že sa môže pripojovať z lokálu. Vzhľadom k tomu, že aj toto je splnené, server umožní užívateľovi pracovať v tejto databáze s definovanými lokálnymi privilégiami. Riaditeľovi boli definované plné lokálne práva k databázi KNIZNICA (a ku všetkým tabuľkám v nej). Takže server príkaz ***use kniznica*** vykoná. Vyskúšame ľubovoľný príkaz v tejto databázi, napr.

mysql> select * from zaner;

a dostaneme očakávaný výsledok, ktorý je vidieť na obr.č.9-4:

```
C:\mysql\bin>mysql -u riaditel -pdirektor
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18 to server version: 3.23.27-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> use mysql;
ERROR 1044: Access denied for user: 'riaditel@localhost' to database 'mysql'
mysql> use kniznica;
Database changed
mysql> select * from zaner;
+---------+--------------+
| cis_odd | tematika     |
+---------+--------------+
|       1 | poezia       |
|       2 | roman        |
|       3 | krimi        |
|       4 | detska lit.  |
|       5 | cestopis     |
|       6 | lit. faktu   |
|       7 | odborna lit. |
+---------+--------------+
7 rows in set (0.33 sec)

mysql>
```

Presvedčíme sa, že užívateľ *riaditel* môže vykonávať ten SQL príkaz, ktorý je definovaný v privilégiách, môže teda vytvárať a mazať tabuľky, napĺňať alebo vyprázdňovať a pod. Vyskúšajme!

Obdobným spôsobom zadefinujeme aj pani vedúcu, slečnu referentku a čitateľa. Ale každému nastavíme iné práva, tak ako sú definované v tab.č.9-6:

Tab.č.9 - 6: **Prístup ostatných užívateľov**

host	user	Select_Priv	Insert_Priv	Update_Priv	Delete_Priv	Create_Priv	Drop_Priv	Reload_Priv	Shutdown_Pri	Process_Priv	File_Priv	Grant_Priv	References_Pr	Index_Priv	Alter_Priv
"localhost"	"veduca"	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	N
"localhost"	"referent"	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N
"localhost"	""	Y	N	N	N	N	N	N	N	N	N	N	N	N	N

Zase sa konektneme ako *root* (lebo len *root* môže pridávať užívateľov!) a zadáme:

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'veduca', 'Y','Y','Y', 'Y','N','N', 'N','N','N','N');

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'referent', 'Y','Y','N', 'N','N','N', 'N','N','N','N');

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', '', 'Y','N','N', 'N','N','N', 'N','N','N','N');

Už azda nemusím pripomínať potrebu „flušu“. Na skušku sa pripojíme ako anonymous. Ako? No predsa príkazom bez parametrov. Taktiež si môžeme zvoliť ľubovoľné meno, napr.

mysql -u billy

Po konektovaní použijeme databázu kniznica. Keďže aj anonymous je definovaný v tabuľke DB, zmenu databáze umožní. Zadáme príkaz ***select***, a aj ten sa vykoná. Ale teraz skúsime pridať záznam do tabuľky ZANER. Čo sa stane? V tabuľke DB je pri userovi “” (anonymous) právo na zápis do tabuľky (Insert_Priv=“N“) zakázané! Takže server túto požiadavku odmietne, čo potvrdzuje aj obr.č.9-5:

```
C:\mysql\bin>mysql -u billy
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21 to server version: 3.23.27-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> use kniznica;
Database changed
mysql> select * from zaner;
+---------+--------------+
| cis_odd | tematika     |
+---------+--------------+
|       1 | poezia       |
|       2 | roman        |
|       3 | krimi        |
|       4 | detska lit.  |
|       5 | cestopis     |
|       6 | lit. faktu   |
|       7 | odborna lit. |
+---------+--------------+
7 rows in set (0.00 sec)

mysql> insert into zaner values(0,'eroticka lit.');
ERROR 1044: Access denied for user: '@localhost' to database 'kniznica'
mysql>
```

Podobne si môžeme overiť práva užívateľov *veduca* a *referent.* Ich možnosti sú prene definované v tab.č.9-6.

Kontrola privilégií
Ako jednoducho overiť, kto má aké privilégiá? No administrátori pod Linuxom dostali do vienka veľmi užitočnú utilitku - **mysqlaccess.** Je to perlovský skript a bohužiaľ, neexistuje na platforme WinXX.
Silu tohoto skriptu si ukážeme inokedy.

Tajomstvo
Tááák, konečne sme sa prehrýzli skutočne najťažšou časťou administrácie MySQL serveru. Bez úplného pochopenia činnosti grant tabuliek sa nedá seriózne pokračovať v tvorbe aplikácií, postavených na SQL serveri. Po dlhých hodinách, čo sme si skúšali rôzne varianty práv, prezradím sladké tajomstvo - dá sa to aj jednoduchšie! A to pomocou SQL príkazu **GRANT** a **REVOKE**. Ale aby sme ich mohli skutočne efektívne používať, musíme dokonale porozumieť činnosti grant tabuliek. No a to vďaka dnešnej kapitole už vieme!

Nabudúce si ukážeme príkazy **GRANT** a **REVOKE.** Tým administrátorská časť seriálu skončí a pustíme sa do omnoho príjemnejšej činnosti - tvorbe aplikácií. Krásnych, oknoidných, nad ktorými aj srdce obyčajného užívateľa zaplesá radosťou. No, veď skúste donútiť vašu účtovníčku pracovať v príkazovom riadku!
Apropó, čo vám hovoria pojmy Apache, PHP, ODBC?

Malé veľké databázy /10.časť

V minulej časti sme si vysvetlili, ako sa definujú globálne a lokálne prístupové práva. Sľúbil som, že si dnes ukážeme omnoho jednoduchší a efektívnejší spôsob. Asi sa teraz pýtate, prečo som vám to neukázal hneď, ale som vás nechal trápiť sa s nejakými grant tabuľkami. No, aby sme rozumeli, čo vlastne konáme, bola prepotrebná predchádzajúca teória. A bez nej by sme neboli schopní analyzovať, čo ktorý užívateľ môže alebo nie. (A takáto analýza sa robí véééľmi často! Veď uvidíte!) Ostatné dôvody akosi vyznejú v dnešnej časti.

Trocha teórie

Vieme, že ak chceme definovať globálne práva k celému SQL serveru, upravujeme tabuľku **USER**. Ak definujeme lokálne práva k niektorej databáze, upravujeme tabuľku **DB** (my dokážeme dokonca definovať práva k jednotlivej tabuľke alebo stĺpcu tabuľky, vtedy upravujeme tabuľky **Tables_Priv** a **Columns_Priv**. Ako sme si však povedali, toto si vysvetlíme inokedy, zatiaľ stačí ten základ - tabuľky **USER**, **DB** a **HOST**).
Ak chceme definovať práva ešte podrobnejšie, upravujeme aj tabuľku **HOST**. Po potrebnej úprave musíme grant tabuľky reflušnúť.
Od verzie 3.23.xx je možné na definíciu prístupových práv použiť príkazy ***GRANT*** a ***REVOKE***. Tieto obdobným spôsobom upravia grant tabuľky v databáze **mysql**, ako sme to my robili ručne v predchádzajúcej časti seriálu.

GRANT
Príkaz ***GRANT*** umožňuje pridávanie nových užívateľov a definovanie prístupových práv nových užívateľov alebo už existujúcich užívateľov.
Syntaktický zápis príkazu ***GRANT*** je takýto:

***GRANT* privilégiá *ON* čo *TO* užívateľ *IDENTIFIED BY* "heslo" [*WITH GRANT OPTION*]**

Pozrime sa na jednotlivé parametre príkazu:

Privilégiá
Tu definujeme privilégiá, ktoré chceme danému užívateľovi prideliť. Môže to byť len jedno privilégium, alebo dve - tri a viac alebo všetky privilégiá. Jednotlivé privilégiá oddeľujeme čiarkou.
V tabuľke č. 10-1 sú popísané jednotlivé privilégiá a ich význam, čo umožňujú. Isto sa vám budu zdať mnohé z nich veľmi podobné privilégiám, ktoré sú definované v grant tabuľkách.

Tab. č. 10-1: Definície privilégií v príkazoch GRANT a REVOKE

Privilégium	Umožňuje
ALTER	Pozmenenie tabuľky alebo indexy
CREATE	Vytvorenie databázy alebo tabuľky
DELETE	Vymazanie existujúceho záznamu z tabuľky
DROP	Odstránenie databázy alebo tabuľky
INDEX	Vytvorenie alebo zmazanie indexu
INSERT	Vloženie nového záznamu do tabuľky
REFERENCES	nepoužité
SELECT	Vybratie existujúcich záznamov z tabuľky
UPDATE	Úpravu existujúcich záznamov
FILE	Čítanie a zápis súborov na server
PROCESS	Prezretie informácií o serveri
RELOAD	Znovunačítanie grant tabuliek a cache tabuliek
SHUTDOWN	Zastavenie servera
ALL	Všetky práva
USAGE	Iba definovať užívateľa bez práv

Vidíme, že prvá skupina parametrov slúži k manipulácii s databázami a tabuľkami. Druhá skupina je skupina administrátorských práv k SQL serveru. Trietia skupina je špecifická - ***ALL*** znamená všetky vyššie popisované práva a ***USAGE*** slúži na vytvorenie nového užívateľa bez všetkých práv.

čo
definuje, k čomu budú stanovené privilégiá definované, teda ku ktorej databáze alebo tabuľke k nej. V tomto prípade sa využíva nám už známy „bodkový zápis“ - *databáza.tabuľka*.
Napr.: *kniznica.kniha* značí, že konkrétne privilégiá budú nastavené k tabuľke *kniha* v databáze *kniznica*.
Existujú aj špecifické „žolíkové“ zápisy:
- ***.*** - značí všetky tabuľky všetkých databáz, teda všade - globálne privilégiá
- *kniznica.**** - značí všetky tabuľky v databáze *kniznica*.

užívateľ
značí, ktorému užívateľovi budú konkrétne privilégiá nastavené. Jednotlivý užívateľ sa definuje menom a klientom, z ktorého sa môže konektovať na server. Takto teda môžeme definovať užívateľa pod rovnakým menom pristupujúceho z rôznych počítačov. Zápis je veľmi podobný emailovej adrese typu *meno@názov_počítača*, napr.:

- *mior@localhost* značí užívateľa *mior* pristupujúceho z lokálnej konzoly
- *mior@192.168.10.152* značí užívateľa *mior* z IP adresy *192.168.10.152*
- *mior@doma* značí užívateľa *mior* z počítača z názvom *doma*.

Ak zadáme iba meno, ale nedefinujeme klienta, z ktorého sa môže hlásiť, znamená to, ako keby sme definovali *“%”* - teda ľubovoľný klient, napr. :
- *mior*
- *mior@*
- *mior@%*

Ak by MySQL server protestoval, vložíme meno aj klienta do úvodzoviek alebo apostrofov, každého zvlášť, napr. *“mior”@”doma”*.

Ak nezadáme meno, ale len klienta, napr. *“”@localhost*, značí to, že definujeme ľubovoľného užívateľa - známeho ako *anonymous*.

heslo
je heslo, ktorým sa bude daný užívateľ autentifikovať. Heslo uvádzame vždy v úvodovkách alebo v apostrofoch.

WITH GRANT OPTION
Tento voliteľný parameter znamená, že takto definovaný užívateľ môže svoje stanovené práva ku konkrétnej databáze alebo tabuľke preniesť na iného užívateľa. Keďže je táto vlastnosť pomerne nebezpečná, používajme ju len v najnutneších prípadoch.

Ako by sme si teda predstavili vyššie uvedený zápis?
Zvoľme si príklad:

GRANT SELECT, INSERT ON kniznica.zaner TO mior@doma IDENTIFIED BY “newpass”

Skúsme si to preložiť ako:

Prideľ práva na selekt a vkladanie k tabuľke *ZANER* v databáze *KNIZNICA* pre užívateľa *mior* z klientského počítača s názvom *doma*, ktorý sa bude autorizovať nielen svojim menom *mior* ale aj heslom *newpass*.

Zložité? Ale ani nie.

Ak nie je ešte užívateľ *mior* z klientského počítača *doma* definovaný v grant tabuľkách, tak sa automaticky vytvorí. V prípade, že už existuje, tak sa iba vykonajú príslušné zmeny privilégií. V takom prípade už nie je potrebné definovať heslo, a tak celú sekciu ***IDENTIFIED BY*** vynecháme. Samozrejme, predtým definované heslo sa zachováva.
Ak chceme pridať užívateľa, ale mu nechceme (zatiaľ) definovať žiadne práva, použijeme privilégium **USAGE** takto:

GRANT USAGE ON *.* to citatel IDENTIFIED BY "heshes"

Poznámka:
Na rozdiel od ručnej manipulácie s grant tabuľkami NIE JE POTREBNÉ tabuľky reflušovať! Toto zabezpečí príkaz ***GRAND*** (aj ***REVOKE***) sám.

REVOKE
Ak chceme nejakému užívateľovi niektoré (alebo aj všetky) privilégiá odobrať, použijeme príkaz ***REVOKE***. Jeho syntaktický zápis je:

REVOKE privilégiá ON čo FROM užívateľ

Zápis je veľmi podobný príkazu ***GRANT***, akurát sa privilégiá nepridávajú k niekomu (*TO*), ale odoberajú od niekoho (*FROM*).
Príkaz ***REVOKE*** nie je schopný užívateľa z grant tabuľky vymazať, iba mu vie odobrať práva. Preto ak chceme niektorého užívateľa z grant tabuliek vymazať, musíme použiť klasický spôsob, napr.:

mysql> DELETE from USER where user = 'meno_uzivatela' AND host = 'meno_pocitaca';
mysql> FLUSH PRIVILEGES;

Tento príkaz je nám už dobre známy a tak si ho nebudeme vysvetľovať. V tomto prípade však nesmieme zabudnúť grant tabuľky reflušnúť!

Vysvetlili sme si základnú teóriu, ale poďme si to teraz ukázať na príkladoch!

Trochu praxe

Aby sme si mohli precvičiť príkazy ***GRANT*** a ***REVOKE***, nadefinujeme si ďalších činovníkov v našej imaginárnej knižnici. Budú to akoby dvojičky k už definovaným užívateľom, ktorým budeme definovať rovnaké práva ako ich vzorom. Veď zastupiteľnosť alebo duplicita je stále aktuálna v štátnej správe. Rovnaké práva nadefinujeme preto, aby sme si mohli pohľadom do grant tabuliek porovnať, že definovanie práv pomocou ***GRANT***/***REVOKE*** je identické s klasickým zadávaním. (Teda skoro, o tom nižšie).
Takže pán riaditeľ bude mať svojho námestníka (kto nevie, čo to znamená, nech sa spýta svojich rodičov), pani vedúcu bude zastupovať pani účtovateľka a slečnu referentku stará pani kuchárka. A čitateľov bude zastupovať pán Karafiát. A do knižnice sa prihlásil aj mladík menom Chytrý, ale ešte nezaplatil členské.
Aha, ešte nám tu chýba jeden veľmi dôležitý človek. Bude to administrátor, pomenujme si ho ***admin***. A prisúdime mu najvyššie správcovské práva. To preto, aby aj root mohol ísť spokojne na dovolenku...
Aby sme mohli overiť ich práva, budeme predpokladať, že všetci pracujú na lokále.

Spomeňme si, aké prístupové práva majú nadefinované pán riaditeľ, pani vedúca, slečna referentka a ľubovoľný čitateľ.
Teraz príkazom ***GRANT*** vytvoríme pána administrátora ako aj jednotlivých zástupcov a zároveň im priradíme zodpovedajúce privilégiá.
Vieme, že administrátor ***admin*** s heslom *super* dostane superužívateľské, teda globálne práva. Preto ich definujeme ako ****.****. Potom príkaz ***GRANT*** vyzerá takto:

mysql> GRANT ALL ON *.* TO admin@localhost
IDENTIFIED BY 'super';

Čo vlastne tento príkaz vykoná? Popíšme si ho „manuálnym" klasickým spôsobom.

mysql> insert into USER (host, user, password,
select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, Reload_priv, shutdown_priv,
process_priv, file_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','admin',password('super'),
'Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y','Y','Y', 'Y','Y');

A aby sa aktivovali zmeny, musíme zadať:

mysql> Flush Privileges;

Teraz zadefinujeme pána námestníka, ktorý sa bude hlásiť heslom *'zastup'*. Tento však už nemá globálne práva, ale len lokálne práva ku všetkým tabuľkám v databáze *KNIZNICA*:

mysql> GRANT ALL ON kniznica.* TO namestnik@localhost
IDENTIFIED BY 'zastup';

Ako by sme to urobili ručne, keby neexistoval príkaz **GRANT**?
Naprv by sme ho mali zadefinovať v tabuľke **USER**, ale nepridelili by sme mu žiadne globálne práva:

mysql> insert into USER (host, user, password,
select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, Reload_priv, shutdown_priv, process_priv, file_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','namestnik',password('zastup'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

Teraz v zmysle minulej časti by sme museli definovať lokálne práva , a to v tabuľke **DB**:

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'riaditel', 'Y','Y','Y', 'Y','Y','Y', 'Y','Y','Y','Y');

Nesmieme zabudnúť na:

mysql> Flush Privileges;

Vidíme, koľko práce nám ušetrí príkaz ***GRANT***!

Podobným spôsobom definujeme pani účtovateľku *ucto* s heslom *financ*:

mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON kniznica.* TO ucto@localhost
IDENTIFIED BY 'financ';

a veľmi obdobne by to v manuálnom zadávaní vyzeralo takto:

mysql> insert into USER (host, user, password,
select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, Reload_priv, shutdown_priv, process_priv, file_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','ucto',password('financ'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'ucto', 'Y','Y','Y', 'Y','N','N', 'N','N','N','N');

mysql> Flush Privileges;

Teraz vytvoríme pani kuchárku, ktorá si zvolila priliehavé heslo *varecha*:

mysql> GRANT SELECT, INSERT ON kniznica.* TO kucharka@localhost
IDENTIFIED BY 'varecha';

Teraz už tušíte, že tento príkaz v skutočnosti vykoná túto trojicu príkazov:

mysql> insert into USER (host, user, password,
select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, Reload_priv, shutdown_priv, process_priv, file_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kuchárka',password('varecha'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'kucharka', 'Y','Y','N', 'N','N','N', 'N','N','N','N');

mysql> Flush Privileges;

Nesmieme zabudnúť na pána Karafiáta s heslom *klincek*, ktorý môže, tak ako anonymous iba prezerať záznamy (select):

mysql> GRANT SELECT ON kniznica.* TO karafiat@localhost
IDENTIFIED BY 'klincek';

A čo sa v skutočnosti vykoná?

No predsa:

mysql> insert into USER (host, user, password,
select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, Reload_priv, shutdown_priv, process_priv, file_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','karafiat',password('klincek'),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'karafiat', 'Y','N','N', 'N','N','N', 'N','N','N','N');

mysql> Flush Privileges;

A nakoniec definujeme mladíka Chytrého. Keďže ani po opakovanej výzve nezaplatil členské príspevky, nepridelíme mu zatiaľ žiadne privilégium a ani heslo:

mysql> GRANT USAGE ON kniznica.* TO chytry@localhost;

Aj u neho by to bolo ručne pracné:

mysql> insert into USER (host, user, password,
select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, Reload_priv, shutdown_priv, process_priv, file_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','chytry',password(''),
'N','N','N', 'N','N','N', 'N','N','N', 'N','N','N', 'N','N');

mysql> insert into DB (host, db, user, select_priv, insert_priv, update_priv, delete_priv, create_priv, drop_priv, grant_priv, references_priv, index_priv, alter_priv)
VALUES ('localhost','kniznica', 'chytry', 'N','N','N', 'N','N','N', 'N','N','N','N');

mysql> Flush Privileges;

(Asi vás napadne jedna vec: Ak príde pán Chytrý a prihlási sa ako *chytry*, nebude môcť vykonať nič, okrem spojenia. Ale keďže je chytrý, prihlási sa ako *anonymous*, teda vymyslí si ľubovoľné meno, ktoré v knižnici

definované nie je, bude môcť aspoň selektovať ! Tu vidíme zdárny príklad bezpečnostnej diery! Alebo sú to tzv. „zadné dvierka“ ?).

Porovnanie
Teraz si porovnáme pridelené globálne prístupové práva, ktoré sme pridelili príkazom ***GRANT*** s tými, ktoré sme pridelili klasickým spôsobom v minulej časti.
Na obr. č.10-1 je výpis tabuľky **USER**. (Tabuľky sú opticky upravené, aby neboli veľmi rozsiahle. Jednotlivé stĺpce zodpovedajú konkrétnym právam):

Host	User	Password	Sel	Ins	Upd	Del	Crea	Drop	Rel	Shut	Proc	File	Grant_	Ref	Index	Alte
localhost	root	4e633cf914a735a0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	riaditel	13a67cae6004a898	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	veduca	3180b21079430047	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	referent	490536835dfc2b19	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost			N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	karafiat	79307ecf29978367	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	kucharka	7045d6a002cdcd34	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	admin	60c033095644bd16	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
localhost	ucto	1f71a07d57e37061	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	namestnik	6c42045d5312478f	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	chytry		N	N	N	N	N	N	N	N	N	N	N	N	N	N

Porovnajme vytvorené dvojice:

- *riaditel - namestnik*
- *veduca - ucto*
- *referent - kucharka*
- *“” (anonymous) - karafiat*

V tomto prípade vidíme, že nemajú pridelené žiadne globálne práva.

Pozrime sa teraz na dvojicu *root - admin*. Obidvaja majú nastavené globálne práva. Sú skoro rovnaké, až na jedno právo - ***Grant_Privileges***. *Root* ho má povolený, *admin* nie. To preto, že pri jeho definícii sme nepoužili parameter ***WITH GRANT OPTION***. V takom prípade *admin* nemôže odovzdať svoje práva niekomu inému. Ako sa vlastne odovzdávajú také práva? No predsa tak, že ten, kto má túto možnosť povolenú, použije príkaz ***GRANT***. Tak prenesie práva na inú osobu. Môžu to byť všetky, alebo len niektoré práva.

Ako vidíme, pán Chtrý nemá páru. A ani on nemá pridelené žiadne globálne práva.

Na obr.č.10-2 je výpis tabuľky **DB**, kde sú definované lokálne práva:

Host	Db	User	Select	Insert	Update	Delete	Create	Drop	Grant	Refere	Index	Alte
localhost	kniznica	riaditel	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	kniznica	veduca	Y	Y	Y	Y	N	N	N	N	N	N
localhost	kniznica	referent	Y	Y	N	N	N	N	N	N	N	N
localhost	kniznica		Y	N	N	N	N	N	N	N	N	N
localhost	kniznica	kucharka	Y	Y	N	N	N	N	N	N	N	N
localhost	kniznica	ucto	Y	Y	Y	Y	N	N	N	N	N	N
localhost	kniznica	namestnik	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
localhost	kniznica	karafiat	Y	N	N	N	N	N	N	N	N	N
localhost	kniznica	chytry	N	N	N	N	N	N	N	N	N	N

Porovnajme naše dvojice:

Vidíme, že *riaditel* a *namestnik* majú rovnaké práva až na ***Grant_Privileges***. To preto, lebo sme ani v tomto prípade nepoužili parameter ***WITH GRANT OPRION***.

Veduca a *ucto* majú tiež rovnaké práva a plne zodpovedajú naším požiadavkám. Tak isto aj *referent* a *kucharka* alebo *anonymous* a *karafiat*.
Len pán *chytry* nemá povolené žiadne právo, pokým nezaplatí členský poplatok.

V tejto tabuľke nie je ani zmienka o *rootovi* a *adminovi*. My už vieme, prečo je to tak. Ak má niekto definované globálne práva (v tabuľke **USER**), už nepotrebuje lokálne práva (v tabuľke **DB**). Globálne práva sú nadradené lokálnym právam.

Ak sme dávali pozor, všimli sme si, že príkazy ***GRANT*** a ***REVOKE*** neovplyvňujú tabuľku **HOST**, ktorá slúži na jemnejšie definovanie prístupových práv. Preto ak potrebujeme takéto členenie, musíme použiť klasické definovanie práv v tejto tabuľke. Avšak v mnohých prípadoch si vystačíme s vyššie použitými postupmi.
Za domácu úlohu si vyskúšajte konektnutie ako jednotliví pracovníci knižnice a skúšajte, čo všetko je vám v databáze knižnica dovolené.
Tým by sme prvý diel seriálu doviedli do konca. Nabudúce sa začneme venovať trochu efektívnejšej práci - tvorbe klientského, užívateľsky príjemnejšieho prostredia.

Miroslav Oravec