Jednosouborový databázový klient. Databáze typu klient-server. Část 2.

Single file database client.
Client-server database. Part 2.

Bc. Eduard Zeman

Diplomová práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Eduard ZEMAN**
Osobní číslo:	**A09525**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Jednosouborový databázový klient. Databáze typu klient-server. Část 2.**

Zásady pro vypracování:

1. **Navyšte výkon, funkčnost a komfort obsluhy existujícího klienta.**
2. **Zpracujte problematiku víceuživatelského přístupu do databáze.**
3. **Teoreticky i prakticky vysvětlete databázové transakce.**
4. **Uveďte možnosti ochrany dat.**
5. **Teoreticky popište a zhodnoťte optimalizaci databáze a klienta.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **STEPHENS, Ryan; PLEW, Ron; JONES, Arie D. Naučte se SQL za 28 dní . [s.l.] : Computer Press, únor 2010. 728 s. ISBN 978-80-251-2700-1.**
2. **OPPEL, Andy. SQL bez předchozích znalostí. Dotisk. [s.l.] : Computer Press, říjen 2008. 260 s. ISBN 978-80-251-1707-1.**
3. **NORTHCUTT, Stephen, et al. Bezpečnost počítačových sítí : Kompletní průvodce návrhem, implementací a údržbou zabezpečené sítě. [s.l.] : Computer Press, říjen 2005. 592 s. ISBN 80-251-0697-7.**
4. **SWAN, Tom. Mistrovství v delphi 4 : Kompletní průvodce pro tvorbu aplikací. Pavel Machek, David Hanousek, Luděk Hořčička. 1. vyd. Praha : Computer Press, 1999. 830 s. Programování. ISBN 80-7226-173-8.**
5. **KOFLER, Michael . Mistrovství v MySQL 5 : Kompletní průvodce webového vývojáře. 1. vyd. Brno : Computer press, 2007. 808 s. ISBN 978-80-251-1502-2.**
6. **GILMORE, W. Jason. VELKÁ KNIHA PHP a MYSQL 5 : kompendium znalostí pro začátečníky i profesionály. 2. přeprac. vyd. [s.l.] : Zoner Press, 2005. 864s. 323-331. ISBN 80-86815-53-6.**
7. **ZEMAN, Eduard. Jednosouborový databázový klient. : Databáze typu klient-server. . [s.l.], 2009. 65 s. Bakalářská práce. UTB, Inženýrská informatika. Vedoucí bakalářské práce RNDr. Ing. Miloš Krčmář.**

Vedoucí diplomové práce: **RNDr. Ing. Miloš Krčmář**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **24. února 2011**

Termín odevzdání diplomové práce: **18. května 2011**

Ve Zlíně dne 24. února 2011

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Zaměřením této diplomové práce je databázový stroj MySQL. Dále pak klient, který byl pro komunikaci s ní naprogramován v předcházející bakalářské práci. MySQL je víceuživatelská relační databáze. Popíšeme blíže její schopnosti. Především se zaměříme na navazování spojení mezi klientem a databází MySQL, z pohledu více než jednoho uživatele. Přiblížíme detailně autentizaci a autorizaci uživatelů. Podrobně se budeme věnovat databázovým transakcím a jejich dopadu na složitost databázového programu. Dále budeme řešit zabezpečení přenosového kanálu proti odposlechu a přineseme také několik informací o zvyšování výkonu databáze. Většina zmiňovaných technologií bude prakticky vyzkoušena a některá z nich pak použita k úpravě klienta.

Klíčová slova: databáze, spojení, zabezpečení, přístupová práva, autentizace, autorizace, transakce, izolace.

ABSTRACT

The asessment of this diploma thesis is a database equipment MySQL and further a client who was programmed in a previous bachelor thesis for the reason of communication with it. MySQL is a multiuser relational database. We will describe its abilities in more detail. Primarily, we will focus on establishing contact between the client and the database MySQL, from more than only one user's point of view. We will closely describe users´ authentication and authorization. We will particularly attend to database transactions and their impact upon the database programme complexity. Furthermore, we will be solving an ensuring of a transmission channel against eavesdropping and we will also bring some information about the database output increase. Most of the hereabove mentioned technologies will be practically proven and one of them will be used for client's adjustment.

Key words: database, contact, ensurance, access right, authentication, authorization, transaction, isolation.

Chtěl bych poděkovat panu RNDr. Ing. Miloši Krčmářovi, za jeho cenné rady a připomínky při tvorbě této diplomové práce. Dále bych chtěl poděkovat své rodině a přátelům za jejich pomoc. Bez nich by to sice šlo také, ale velice, velice těžce. Neboť jak praví klasik - *„Maličkosti tvoří dokonalost, ale dokonalost není maličkostí.“* Stejně tak i malá pomoc vždy přispěje k jakémukoliv cíli.

Když je těch malých pomocí hodně ……

Děkuji všem.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD **9**

I TEORETICKÁ ČÁST **10**

1 PŘÍPRAVA NA TEORII **11**

1.1 VÍCEUŽIVATELSKÝ PŘÍSTUP DO DATABÁZE 11

1.1.1 Autentizace 13
1.1.2 Založení nového uživatele a test autentizace 24
1.1.3 Autorizace 27
1.1.4 Autentizace a autorizace – shrnutí 31

1.2 TRANSAKCE 34

1.2.1 Jednouživatelské transakce 35
1.2.2 Transakce ve víceuživatelském prostředí 41
1.2.3 Transakce - příklad 1 42
1.2.4 Transakce - příklad 2 43
1.2.5 Izolace transakcí 45
1.2.6 Křížení transakcí 46
1.2.7 Křížení transakcí – příklad 1 46
1.2.8 Křížení transakcí – příklad 2 48
1.2.9 Transakce – shrnutí 49

1.3 ZABEZPEČENÍ 49

1.3.1 Bezpečnost serveru 50
1.3.2 Bezpečnost spojení 51
1.3.3 OpenSSL – vytvoření certifikátu 53
1.3.4 Bezpečnost klienta 57

1.4 NAVYŠOVÁNÍ VÝKONU MYSQL 57

1.4.1 HW navýšení výkonu 59
1.4.2 Navýšení výkonu stávajícího HW 59
1.4.3 Rozprostření výkonu na více počítačů 62

II PRAKTICKÁ ČÁST **64**

2 VÝBĚR POTENCIALNÍCH ÚPRAV **65**

2.1 VÍCEUŽIVATELSKÝ PŘÍSTUP DO DATABÁZE 66

2.2 AUTENTIZACE KLIENTA NA ÚROVNI DATABÁZOVÉHO STROJE MYSQL 67

2.3 AUTENTIZACE KLIENTA NA ÚROVNI DATABÁZOVÉ TABULKY AUDIO_VIDEO 69

3 MODIFIKACE KLIENTA 71
3.1 MODIFIKACE PŘÍSTUPOVÝCH PRÁV MYSQL 71
3.2 ÚPRAVA KLIENTA – AUTOMATICKÉ PŘIHLÁŠENÍ DO REŽIMU SELECT 73
3.3 ÚPRAVA KLIENTA – PŘIHLÁŠENÍ A REGISTRACE 75
3.4 ÚPRAVA KLIENTA – KÓD REGISTRACE 78
3.5 ÚPRAVA KLIENTA – KÓD PŘIHLÁŠENÍ 79
3.6 ÚPRAVA KLIENTA – KONTROLA EXPIRACE ÚČTU 80
3.7 ÚPRAVA KLIENTA – ODCHYCENÍ CHYBOVÝCH HLÁŠENÍ 81
3.8 ÚPRAVA KLIENTA – VÍCEUŽIVATELSKÝ PROVOZ 82
3.9 TRANSAKCE 83
3.10 BEZPEČNÉ PŘIPOJENÍ 87
ZÁVĚR 89
CONCLUSION 90
SEZNAM POUŽITÉ LITERATURY 91
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 93
SEZNAM OBRÁZKŮ 94
SEZNAM TABULEK 97
SEZNAM PŘÍLOH 98

ÚVOD

Databázové programy se používají téměř všude. Na úřadech, u lékaře, na Internetu a také třeba v mobilních telefonech. O jejich funkčnosti je potřeba vědět co nejvíce informací, aby mohly fungovat kvalitně, bezproblémově a hlavně bezpečně.

Databáze MySQL je velice často provozována na webových serverech s elektronickými obchody. Na tyto weby přistupuje ve stejný okamžik mnoho zákazníků, kteří mají často shodné požadavky. Databáze je musí všechny bezchybně obsloužit. Prvním cílem této diplomové práce bude pochopit její klíčové vlastnosti. Zaměříme se nejprve na identifikaci zákazníků. Z pohledu databáze MySQL nás bude zajímat autentizace a autorizace uživatelů. Poté se budeme věnovat problematice zpracování většího množství požadavků ve stejný časový okamžik. Rozebereme transakce a jejich izolace. Prozkoumáme jednu z metod obrany proti odposlechu komunikace mezi klientem a databází. Nakonec si popíšeme možnosti navyšování výkonu databáze.

Tato diplomová práce navazuje na bakalářskou práci [8] a rozšiřuje ji. V bakalářské práci jsme si dali za cíl naprogramovat databázového klienta. Úkolem klienta bylo připojit se na vzdálenou databázi MySQL, na ní pak manipulovat s daty a samozřejmě data také tisknout. Rovněž své vlastní konfigurace musel být schopen ukládat do databáze a to vše s podmínkou, že bude zkompilován do jediného, přímo spustitelného souboru.

Druhým cílem diplomové práce tedy bude integrace některých výše uvedených technologií do tohoto klienta. Naprogramujeme registrace, přihlašování uživatelů a změnu jejich privilegií. To znamená, že se klient bude muset dělit o jednu databázi ještě s jinými přistupujícími klienty. Pokud to bude nutné, naučíme klienta i transakce a zabezpečovat spojení.

I. TEORETICKÁ ČÁST

1 PŘÍPRAVA NA TEORII

Teorie je krásná a užitečná věc. Pro mnoho lidí však může znamenat značnou otravu. Zejména technický popis problému, který je možné prakticky prezentovat za 5 minut, může trvat hodně dlouho. A ještě nemusí být správně pochopen. Důležitý je i kontext. Snad to zní malicherně, ale pokud se něco učíme a pochopíme to, tak zjistíme, že si to nemusíme pamatovat. Právě proto, že jsme pochopili kontext. Naopak informace, které jsme si freneticky zapamatovali, aniž bychom pochopili jejich podstatu, se nám zakrátko vykouří z hlavy. Ani ne tak zakrátko, jako spíš hned. Proto pojmeme teoretickou část následujícím způsobem. Povíme si něco málo o daném problému a pak ho aplikujeme na nějaké malé testovací sadě dat. Co nám neřekne teorie, to nám řekne praxe.

1.1 Víceuživatelský přístup do databáze

Existují databáze, které provozujeme na svém **PC**. Klient i databáze mohou být implementováni do jediné aplikace. Tato možnost sice existuje, ale není příliš častá. Jinou možností je použít klienta a databázi odděleně. Stále však na stejném PC. Konečně pak databáze a klienti, kteří jsou odděleni počítačovou sítí. Ve všech případech musíme řešit problém s uživateli, kteří se do těchto databází přihlašují. Položme si otázku, proč?

Začneme domácími aplikacemi typu *„můj denní stravovací režim a osobní deníček“*. Do této databáze se přihlašují otec, matka, dvě dcery a dva kluci. Předpokládáme, že všechny děti jsou v rozsahu pubertálního věku. Všem je asi jasné, že data uživatelů by měla zůstat striktně oddělená a chráněná minimálně přístupovým heslem. To, že si dal otec bůček místo ovesných vloček, je jenom jeho věc. Stejně tak děvčata určitě nesvěří své osobní problémy svým bratrům.

Separabilita uživatelských účtů a dat je ve většině případů zřejmá a žádoucí. V některých případech je dokonce kriticky nutná. Například u bankovních účtů. Naopak databáze obecných informací jsou veřejné a není zde nutná žádná autentizace. Typicky **wikipedia** [13]. Obecně se při rozhodování o zařazení autentizačních procedur řídíme mnoha aspekty.

1. Mohou být data z databáze zcela volně přístupná?
2. Jsou data v databázi nějak vázána na konkrétního uživatele?
3. Jsou data v databázi kritického charakteru?
4. Jsou data v databázi indiskrétního charakteru?

Otázek je celá řada. Ve své podstatě záleží vše pouze na jediné otázce. Jsou data, která chceme uložit do databáze, pro nás nějak důležitá a osobní? Pokud ano, je autentizace naprosto nezbytná. Data jsou určená pouze nám a nikomu jinému. Pozor! Nemluvíme nyní o zabezpečení dat na úrovni databázového záznamu, tabulky a celé databáze. Tedy z hlediska souborového systému, ve kterém je databáze fyzicky umístěna. Předpokládáme, že s použitím dostatečně silné autentizace jsou data v bezpečí. O aspektech fyzického zabezpečení dat a databáze budeme hovořit v jiné kapitole. V následující kapitole si řekneme něco o autentizaci a autorizaci. Vytvoříme si také fyzickou ukázku. Od této chvíle se budeme věnovat databázi **MySQL** [12]. Na samotném začátku výkladu si musíme definovat několik důležitých pojmů.

MySQL	je databáze, databázový program či databázový stroj
mysql	je jeden z možných databázových klientů databázového programu MySQL
mysql	je jedna z databází databázového programu MySQL.
mysqld	je jeden z démonů (ovladačů) databázového programu MySQL.

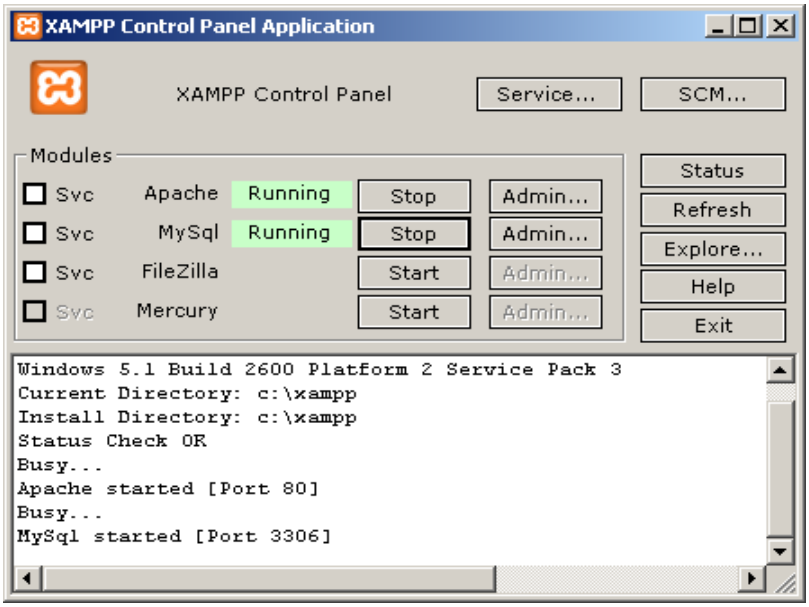
Je to celkem zmatek. Z hlediska funkcionality je to velice dobré řešení. Ostatně to sami uvidíme. Avšak z hlediska popisování se můžeme občas dostat do situace, kdy nebudeme vědět, který z výše uvedených popisů právě používáme. Je to jenom otázka zvyku. Nejedná se o náš výmysl. Výrobce databázového programu MySQL to skutečně takto zamýšlel. Pravidlem bude, že všechny databáze a databázové tabulky budeme psát tučným písmem s kurzívou. Alespoň takto malinko si pomůžeme. U ostatních výrazů se budeme snažit o jednoznačnost.

1.1.1 Autentizace

Autentizace je jednoznačná identifikace osoby. Například podle občanského průkazu nebo cestovního pasu. Osobu, která se prokáže platným dokladem, lze považovat za autentickou. Kriminální živly zanedbáme. V databázovém světě je autentizace prvním krokem k navázání spojení mezi klientem a databázovým strojem. Podle občanky to bohužel není možné. Vlastní autentizace probíhá nejčastěji na základě jména a hesla. Tyto údaje jsou nám přiděleny databází, oprávněnou osobou nebo si je zvolíme sami. Je také možné provádět autentizaci pomocí biometrických informací. Například na základě daktyloskopie.

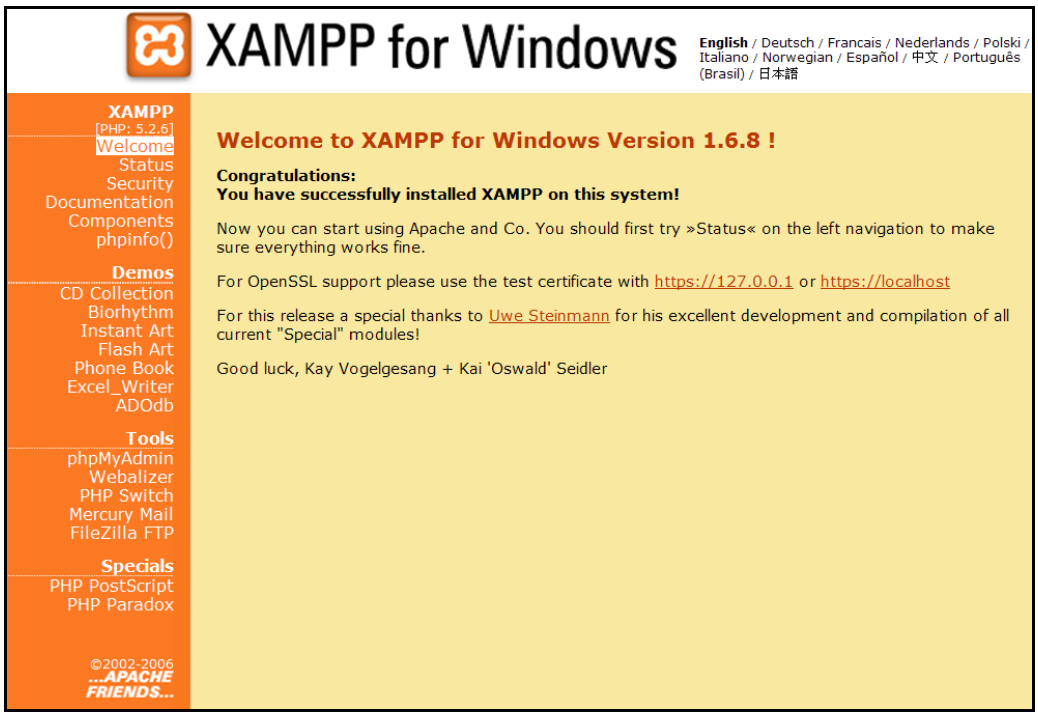
Technologií existuje celá řada. Jméno a heslo je pro většinu aplikací dostatečné. Ne však pro všechny aplikace. Některé ještě vyžadují autentizaci třetí osobou. Říká se jí certifikační autorita. Ta nás nyní nebude zajímat. Je předmětem jiné kapitoly. Vlastní autentizace podle jména a hesla jde dále rozšířit o identifikaci zdroje, který databázi volá. Celá autentizace pak testuje jméno, **IP** adresu volajícího a jeho heslo. To je i náš případ. Stroj MySQL [1], [2], [3], [4] má velmi propracovanou autentizaci a následnou autorizaci uživatelů. Má dvě fáze. V první fázi je vlastní autentizace. Ta nám pouze umožní přístup k databázovému stroji MySQL. V druhé fázi pak probíhá autorizace. MySQL zjišťuje, co všechno smí uživatel s databází provádět. Komunikace klienta se strojem MySQL neprobíhá přímo. Komunikuje se přes systémovou službu, které se říká server. Tato služba je realizovaná démonem nebo chceme-li ovladačem mysqld. Databázový program může mít více různých speciálních ovladačů. Standardně se používá právě mysqld.

Přistupme přímo k našemu databázovému programu MySQL a podívejme se na něj. Nejprve si připomeňme postup zprovoznění MySQL. Tento postup je popsán v bakalářské práci [8]. Potřebujeme webový server **Apache**, interpreter **PHP**, a databázový stroj MySQL. Apache a PHP je potřeba pro program PhpMyAdmin (PMA). PMA je jenom jedním z mnoha klientů MySQL od nezávislého výrobce. Přes tento program se budeme dívat na naše data a na strukturu databází. Ideálním nástrojem pro zprovoznění výše uvedených programů je program XAMPP [10] ve verzi (1.6.8). Tento program obsahuje po nainstalování vše potřebné. Detailní popis instalace je rovněž v bakalářské práci [8]. Na následujícím obrázku (Obr. 1) je spuštěn program XAMPP.



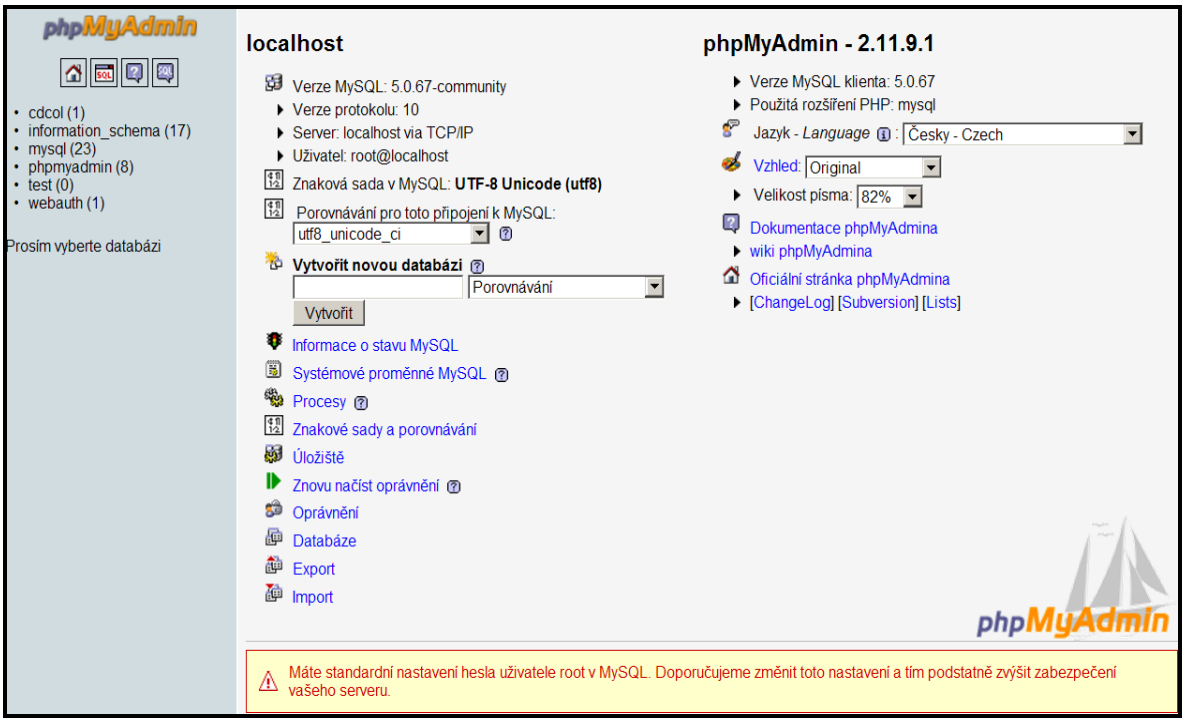
Obr. 1. XAMPP - Control Panel Application.

Stačí nastartovat Apache a MySQL. Poté je třeba spustit internetový prohlížeč. Do řádku pro zápis adresy napíšeme pouze localhost. Prohlížeč si doplní zbytek na http://localhost/. Získáme uvítací stránku XAMPP (Obr. 2).



Obr. 2. XAMPP - uvítací stránka.

Zde vybereme PhpMyAdmin. Získáme grafické rozhraní pro ovládání MySQL (Obr. 3). V levé části máme seznam všech databází, které MySQL po instalaci obsahuje.



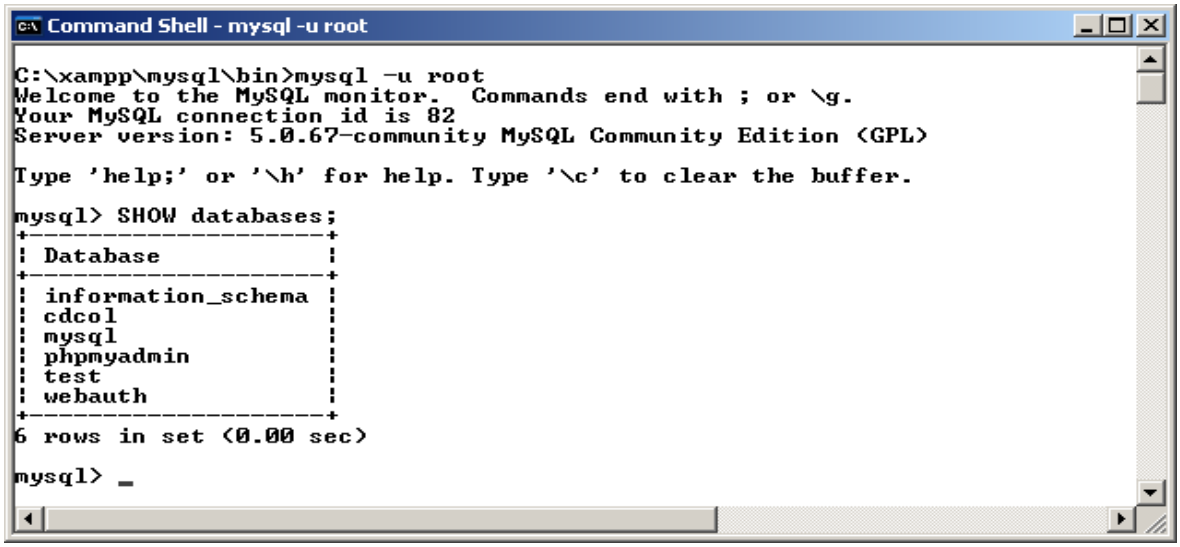
Obr. 3. PhpMyAdmin – seznam databází.

Tedy celkem 6 položek. Primárně nás bude zajímat databáze ***mysql***. Zajímavé přitom je, že struktura databáze ***mysql*** je stejná jako jakákoliv jiná databáze. Ovladač mysqld samozřejmě ví, že právě v této databázi jsou životně důležitá konfigurační data serveru MySQL. Za povšimnutí ještě stojí varování, které nás upozorňuje, že uživatel *root* nemá přiděleno žádné heslo. Toho si zatím nebudeme všímat. Později vše napravíme.

Než přistoupíme k praktické ukázce autentizace, potřebujeme ještě jednoho klienta. MySQL jich má celou řadu. Některé jsou grafické jako PhpMyAdmin. Jiné jsou řádkové. Primárním klientem MySQL je právě klient, který se spouští z běžného příkazového řádku. Tohoto klienta budeme hojně využívat. Pokud jsme program XAMPP instalovali se standardním nastavením, tak ho najdeme v adresáři „c:\xampp\mysql\bin“. Spusťme příkazový řádek v tomto adresáři a vepišme mysql. Program vypíše následující chybu:

```
ERROR 1045 (28000): Access denied for user 'ODBC'@'localhost' (using
password: NO)
```

To je naprosto v pořádku. Program se pokusil spojit se serverem MySQL. Použil přitom defaultní nastavení. To je úmyslně naprosto špatně. Uživatel *ODBC* neexistuje. Program má mnoho specifikačních atributů. Nás budou zajímat atributy *-p* a *-u*. Atribut *-u* definuje uživatele a atribut *-p* pak žádá jeho heslo. Na obrázku (Obr. 4) je ukázka přihlášení na uživatele *root*. Ten zatím nemá žádné heslo.



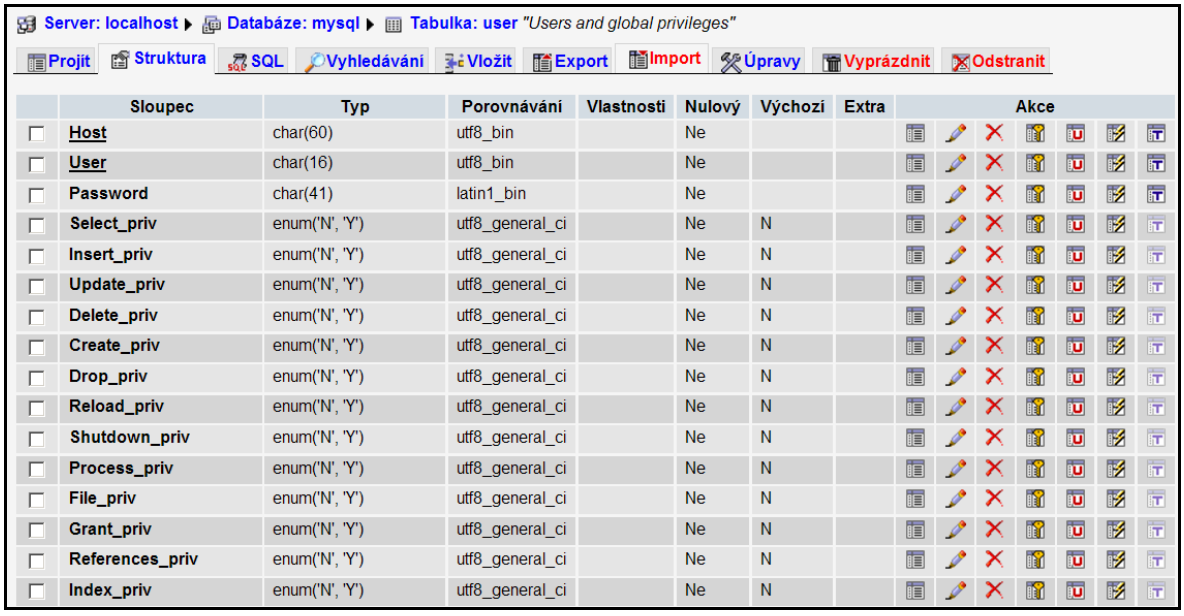
Obr. 4. Mysql - příkazový řádek programu.

Vidíme zde dvě věci. Příkaz „mysql –u root“ provedl spojení klienta mysql s databázovým strojem MySQL. Zatím pouze obecně. Nespecifikovali jsme žádnou konkrétní databázi. Druhý příkaz, „SHOW databases“, nyní již v rámci spuštěného klienta, provedl výpis všech dostupných databází. Pokud se podíváme na obrázek 3 a 4 uvidíme, že oba klienti vrátili stejná data. Může nás zarazit fakt, že program PhpMyAdmin nepoužívá žádné heslo ani jméno. To proto, že PhpMyAdmin se na server MySQL přihlašuje automaticky jako *root*. Prozatím bez hesla.

Podívejme se nyní na strukturu databáze ***mysql***. Použijeme na to klienta PhpMyAdmin. Je to přece jenom komfortnější. Klikneme na databázi ***mysql***. Dostaneme následující výstup (Obr. 6). Uvedeme jenom část výpisu.

Obr. 6. PhpMyAdmin - přehled struktury databáze mysql.

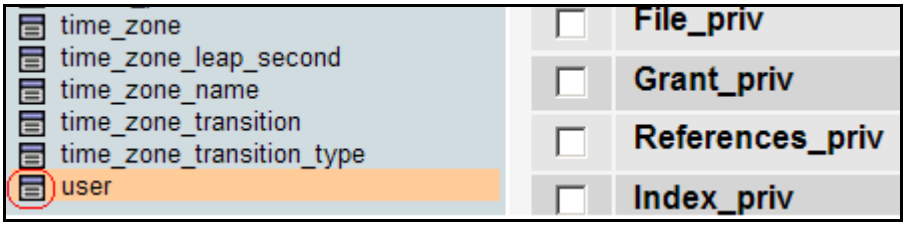
V levém sloupci máme všechny tabulky, které jsou v databázi ***mysql***. Tato databáze má na starosti autentizaci a autorizaci uživatelů. O autentizaci se primárně stará tabulka ***user***. Chceme-li se podívat na strukturu této tabulky, stačí kliknout na její název. Část její rozsáhlé struktury ukazuje obrázek (Obr. 7). Zajímají nás první 3 řádky.



Obr. 7. PhpMyAdmin - část struktury tabulky user.

1. *Host* určuje IP nebo alias počítače, ze kterého se k MySQL přihlašujeme.
2. *User* definuje přihlašovací jméno.
3. *Password* pak přihlašovací heslo.

K autentizaci jsou potřebné i další řádky této tabulky. Například počet maximálního možného přihlášení jednoho uživatele za hodinu. Nebo povoleni **SSL** komunikace. To nás však zatím nezajímá. Hodnoty jsou nastaveny a my ostatní parametry nebudeme používat. Nyní se podívejme, proč se mohl PhpMyAdmin přihlásit bez hesla. Stejně tak uživatel *root*. To uděláme tak, že klikneme na obálku vedle názvu tabulky (Obr. 8).



Obr. 8. PhpMyAdmin - detail obálky.

Nyní konečně získáme konkrétní záznamy z tabulky ***user*** (Obr. 9).

	Host	User	Password	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv
	localhost	root		Y	Y	Y	Y	Y
	localhost	pma		N	N	N	N	N
	127.0.0.1	root		Y	Y	Y	Y	Y

Obr. 9. PhpMyAdmin - data tabulky user.

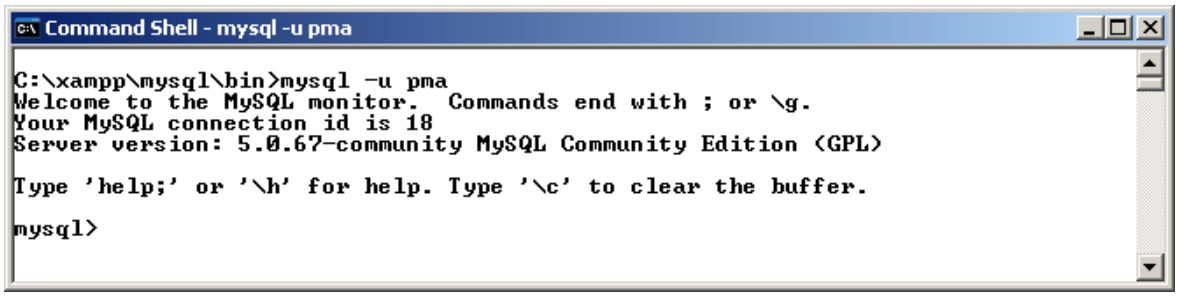
V tabulce jsou 3 záznamy. Dva pro uživatele *root* a jeden pro uživatele *pma*. Sloupeček *Password* neobsahuje žádný záznam. Takže se všichni tři uživatelé mohou připojovat bez hesla. Uživatel *root* se může přihlašovat z počítače „localhost“ a „127.0.0.1“. Jedná se o tentýž přístupový bod. „localhost“ je pouze alias k IP adrese „127.0.0.1“.

Zkusme modifikovat uživatele *pma*. V tabulce klikneme na symbol tužky u tohoto uživatele a dostaneme formulář pro úpravu záznamu (Obr. 10).

Sloupec	Typ	Funkce	Nulový	Hodnota
Host	char(60)			localhost
User	char(16)			pma
Password	char(41)			test

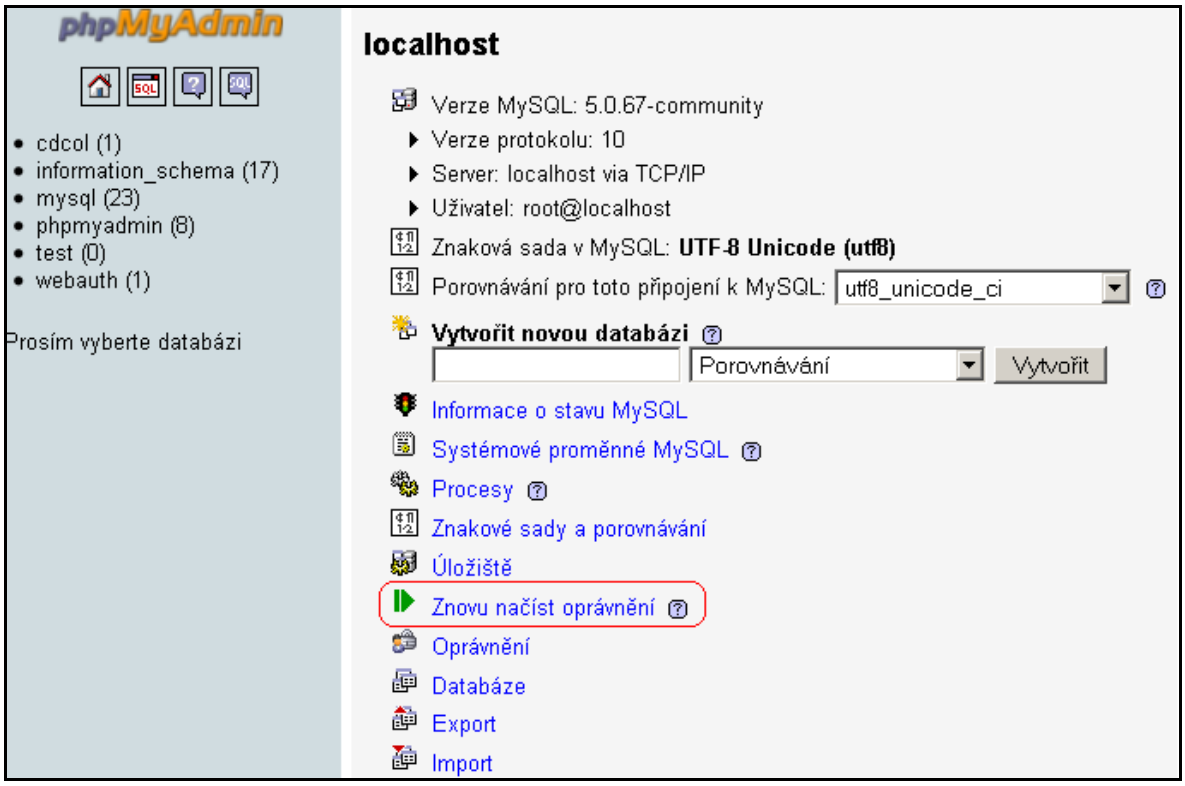
Obr. 10. PhpMyAdmin - modifikační formulář.

Do řádku *Password* si napíšeme heslo „*test*“. Úplně dole na stránce s formulářem je pak tlačítko „proveď“. Tím se nám upravený záznam zapsal do databáze. Nyní se k MySQL přihlásíme přes řádkového klienta mysql. Prostřednictvím uživatele *pma* (Obr. 11). Přihlašovací příkaz „mysql –u pma“ prošel bez žádosti o heslo. Jak je to možné? Toto je nesmírně důležitá věc. Databázový ovladač mysqld má kopie tabulek databáze ***mysql*** načteny v paměti **RAM**. Je tak učiněno pro zvýšení výkonu při autentizaci a následné autorizaci. Náš řádkový klient nemá nejmenší tušení o změně přístupových práv.



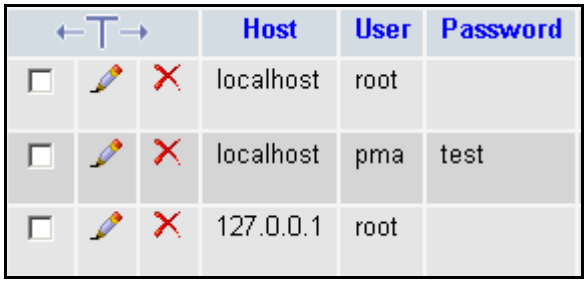
Obr. 11. Mysql - chyba v přihlášení.

Tuto chybu musíme ihned napravit. Je to problém vyrovnávacích pamětí obecně. Přímo v PhpMyAdmin klikneme vlevo nahoře na symbol domečku. Získáme výchozí stránku tohoto programu. Najdeme tam položku „znovu načíst oprávnění“ (Obr. 12).



Obr. 12. PhpMyAdmin - úvodní stránka a reset privilegii.

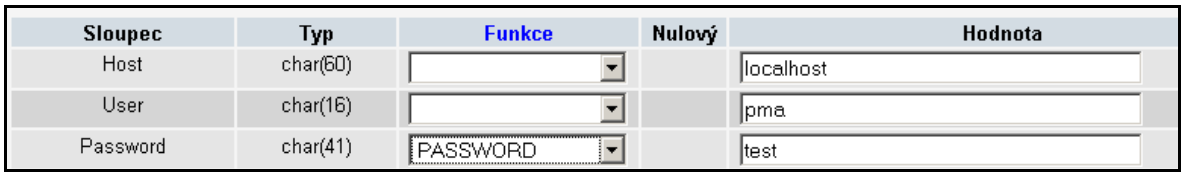
Na pozadí se provede SQL příkaz „FLUSH PRIVILEGES“. Ten znovu načte do RAM opravené kopie tabulek. Vraťme se k řádkovému klientu. Odhlasme se. Při novém pokusu o přihlášení se na uživatele *pma* bez hesla, dojde k chybě. Zkusme to s heslem „*test*“. Zadejme přihlašovací údaje „mysql –u pma –p“. Jsme vyzváni k zadání hesla. Po odeslání dotazu dostaneme opět chybu. Opět jsme neuspěli. To už začíná být podezřelé. Podívejme se znova na výpis tabulky ***user*** v programu PhpMyAdmin (Obr. 13).



Obr. 13. PhpMyAdmin - detail tabulky user.

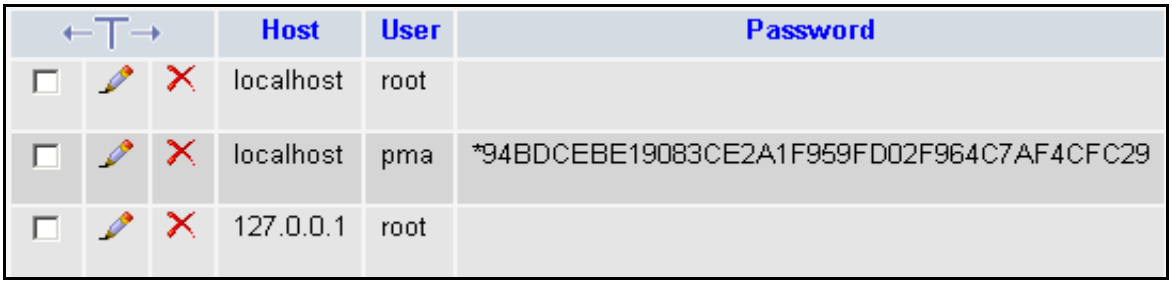
Vidíme, že uživatel *pma* má nastaveno heslo „*test*“. Tak co se to děje? Vysvětlení je prosté. Tabulka ***user*** je zcela normální součástí souborového systému. Jediná možná

ochrana proti zcizení přístupového hesla je jeho zašifrování. Totéž samozřejmě může platit pro ostatní data v jiných tabulkách. Nemusí to být jenom záležitostí hesla. MySQL používá na hesla jednosměrné šifrovací algoritmy. Při modifikaci hesla musíme vybrat funkci PASSWORD. Ta zajistí zašifrování hesla (Obr. 14).



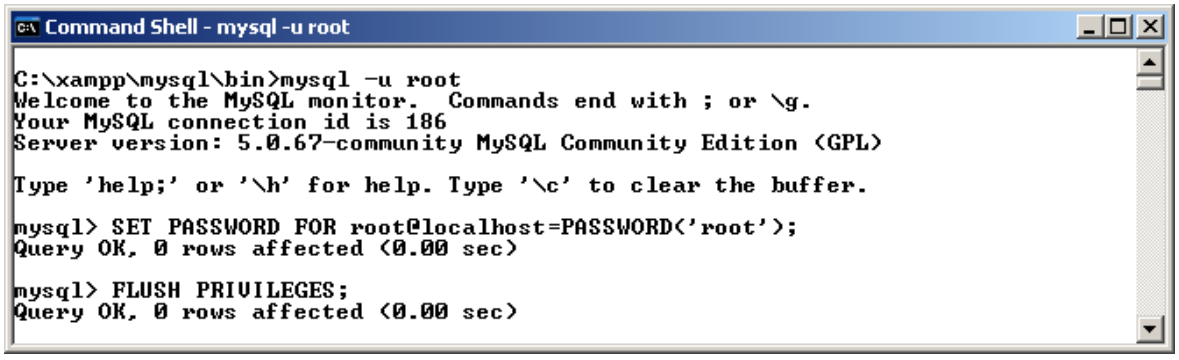
Obr. 14. PhpMyAdmin - aktivace funkce PASSWORD při modifikaci hesla.

Následný výpis tabulky ***user*** (Obr. 15). Výsledek je zcela jiný.



Obr. 15. PhpMyAdmin - zašifrované heslo.

Provedeme znovunačtení oprávnění do RAM a konečně se přihlásíme na účet *pma* i z řádkového klienta. Reset oprávnění může chviličku trvat. Po několika sekundách se nám již bez problému podaří realizovat přihlášení na uživatele *pma*. Nyní se ještě podíváme, jak snadno můžeme měnit hesla v řádkovém klientu. Přihlasme se jako uživatel *root*. PhpMyAdmin nás pořád ještě upozorňuje, že uživatel *root* nemá nastaveno žádné heslo. To je kritická chyba. Uživatel *root* má v tabulce ***user*** nastavena všechna oprávnění na „YES“. Je to tedy takzvaný superuživatel. Ten může naprosto vše. U jeho účtu by mělo být silné heslo. Změníme ho. Provedeme to pomocí řádkového klienta mysql (Obr. 16). Pro demonstrační účely nyní zvolíme jednoduché heslo.



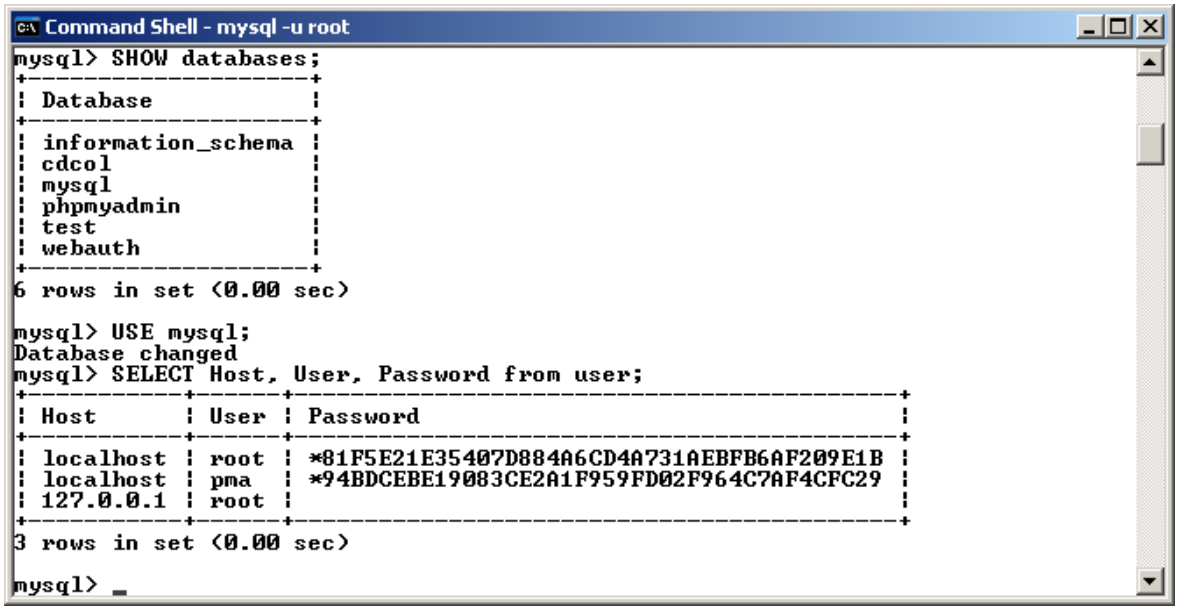
Obr. 16. Mysql - změna hesla.

Velice snadné a rychlé.

```
SET PASSWORD FOR root@localhost=PASSWORD('root');
```

Tento příkaz vložil nové heslo „*root*“ do záznamu, kde User = *root* a Host = *localhost*. Navíc zavolal funkci PASSWORD, která zašifrovala řetězec „*root*“. Potom jsme provedli příkaz „FLUSH PRIVILEGES“. Tím jsme aktualizovali oprávnění v paměti RAM. Určitě stojí zato si nyní prohlédnout záznamy v tabulce ***user***. Udělejme to přes klienta mysql. V programu PhpMyAdmin to již umíme. V řádkovém klientu jsme stále ještě přihlášeni jako *root* bez hesla. Provedeme nové přihlášení uživatele *root*. Nyní je již požadováno heslo. Zadejme ho.

V MySQL může být mnoho databází. Příkazem „SHOW databases“ si můžeme prohlédnout disponibilní databáze (Obr. 17). Mezi nimi je i ***mysql***. Vstoupíme do ní tak, že zavoláme příkaz „USE mysql“. Nyní jsme v databázi ***mysql*** a můžeme s ní začít pracovat.



Obr. 17. Mysql - přehled databází, nastavení pracovní databáze a výpis záznamů.

Příkazem „SELECT Host, User, Password from user“ vypíšeme záznamy z tabulky ***user***. Požadujeme pouze sloupečky *Host*, *User* a *Password*. Záznamy v tabulce ***user*** lze měnit i standardními SQL příkazy. Nesmíme však zapomenout na volání funkce PASSWORD pro uložení hesla v zašifrované podobě. POZOR! Pokud změníte heslo a nepoužijeme funkci PASSWORD, tak si můžete příslovečně podřezat větev, na které sedíme. Vložíme-li například do účtu *root* heslo „*test*“ v nezašifrované podobě, tak už bychom se na tento účet nedokázali přihlásit. Jedině přes jiného super uživatele, který by sjednal nápravu. Pokud takový uživatel v databázi není, znamená to většinou veliké problémy. Vyřešit se to dá například přímou editací souboru tabulky **user**. Smazáním hesla u uživatele *root*. Další možností je zastavit démona mysqld a znovu ho spustit s přepínačem „skip_grant_tables“. Server MySQL bude ignorovat tabulku ***user***. Nebude se tedy provádět autentizace. Přihlásíme se jako *root* bez hesla a modifikujeme heslo. Poté samozřejmě spustíme démona mysqld běžným způsobem. A tady nastává další bezpečnostní problém, o kterém si ještě povíme. Na závěr této kapitoly si stručně shrneme autentizační proces.

Při přihlášení se k databázovému programu MySQL dojde nejprve k prověření oprávnění podle tabulky ***user*** v databázi ***mysql***. Jednoduše řečeno se porovnají dodané insignie se záznamy v tabulce ***user***. Pokud některý řádek vyhovuje dodaným údajům, je přihlášení povoleno. V autentizačním procesu se porovnávají tyto údaje.

1. Z jaké IP adresy se smíme přihlásit.
2. Pod jakým přihlašovacím jménem.
3. S jakým heslem.
4. Dále se testuje požadavek na bezpečné připojení SSL.
5. Nakonec pak ještě limit maximálního počtu povolených přihlášení.

Tato celá kapitola pojednávala o autentizaci uživatelů. Nezbývá nám než otestovat nabyté zkušenosti a nové vědomosti.

1.1.2 Založení nového uživatele a test autentizace

Našim úkolem je vytvořit tři uživatele.

1. Uživatel *Jan* se smí přihlašovat z počítače s IP adresou „212.210.200.250“. Jeho heslo bude „*popelka*“.
2. Dále chceme, aby se z počítače s IP adresou „192.168.2.1“ mohli přihlašovat všichni uživatelé bez hesla.
3. Pak požadujeme vytvoření uživatelky *Dagmar*. Ta se bude moci přihlašovat z počítače „localhost“ a „200.200.200.125“ s heslem „*leden*“.
4. Na závěr pak uživatele *Eda*, který se může pod heslem „*jaro*“ přihlašovat odkudkoliv.

Pro první dva body použijeme klienta PhpMyAdmin. Pro další dva pak mysql. V programu PhpMyAdmin si nastavíme heslo uživatele *root* opět na prázdno. PhpMyAdmin má přihlašovací údaje standardně nastaveny na uživatele *root@localhost* bez hesla. Tyto údaje má ve svém konfiguračním souboru. Je zbytečné je hledat a měnit. V řádkovém klientu si velice rychle nastavíme heslo uživatele *root* na prázdný znak. Bez jedné nebo druhé úpravy, skončí volání programu PhpMyAdmin chybou přístupu. Upravíme heslo:

```
SET PASSWORD FOR root@localhost=PASSWORD('');
```

Založíme nyní všechny nové uživatele.

1. PhpMyAdmin a uživatel *Jan* (Obr. 18). Nezapomeňme na funkci PASSWORD.

Sloupec	Typ	Funkce	Nulový	Hodnota
Host	char(60)			212.210.200.250
User	char(16)			Jan
Password	char(41)	PASSWORD		popelka

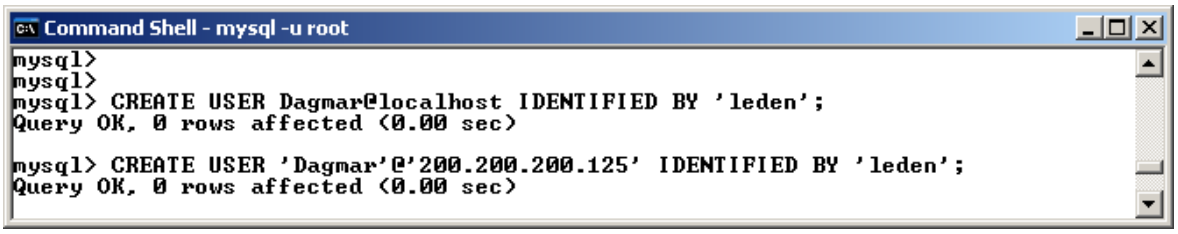
Obr. 18. PhpMyAdmin - vytvoření uživatele Jan.

2. PhpMyAdmin a uživatel bez přihlašovacího jména (Obr. 19). Obecně je používání takovýchto přístupů nežádoucí. Jsou příliš otevřené z bezpečnostního hlediska.

Sloupec	Typ	Funkce	Nulový	Hodnota
Host	char(60)			192.168.2.1
User	char(16)			
Password	char(41)			

Obr. 19. PhpMyAdmin – vytvoření obecného uživatele.

3. Mysql a uživatelka *Dagmar*. Přístup ze dvou počítačů (Obr. 20).



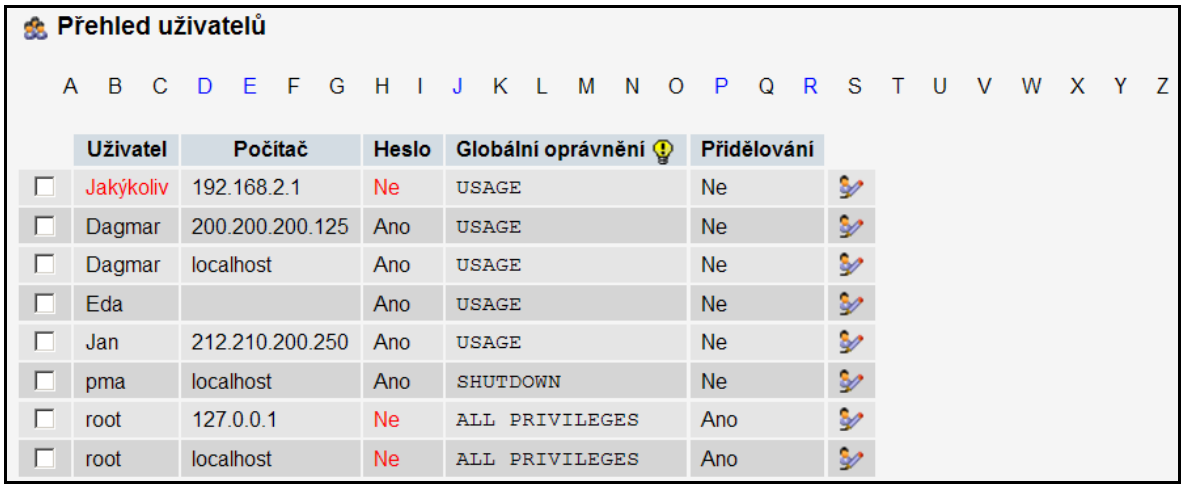
Obr. 20. Mysql - vytvoření uživatele Dagmar.

4. Mysql a uživatel *Eda*. Přístup odkudkoliv (Obr. 21). V MySQL lze definovat obecný přístup také znakem „%". Pak by vypadal zápis uživatele takto:

```
CREATE USER 'Eda'@'%' IDENTIFIED BY 'jaro';
```

Obr. 21. Mysql - vytvoření uživatele Eda.

Celkový pohled na vytvořené uživatele (Obr. 22). PhpMyAdmin, domovská stránka, oprávnění. I takto se dá získat přehled o uživatelích.



	Uživatel	Počítač	Heslo	Globální oprávnění	Přidělování	
☐	Jakýkoliv	192.168.2.1	Ne	USAGE	Ne	
☐	Dagmar	200.200.200.125	Ano	USAGE	Ne	
☐	Dagmar	localhost	Ano	USAGE	Ne	
☐	Eda		Ano	USAGE	Ne	
☐	Jan	212.210.200.250	Ano	USAGE	Ne	
☐	pma	localhost	Ano	SHUTDOWN	Ne	
☐	root	127.0.0.1	Ne	ALL PRIVILEGES	Ano	
☐	root	localhost	Ne	ALL PRIVILEGES	Ano	

Obr. 22. PhpMyAdmin - seznam uživatelů a jejich globální privilegia.

Na seznamu lze vidět, že některé položky jsou červeně. Ty nám připomínají, že je něco celkem špatně. Potenciální bezpečnostní riziko. Typicky nepřítomnost hesla, či příliš univerzální přístup. V tabulce dále vidíme sloupeček, ve kterém je globální oprávnění. Uživatel *root* může všechno. Uživatel *pma* může zastavit server MySQL. Ostatní nemohou zatím vůbec nic. Tuto skutečnost si ukážeme v následující kapitole.

S přístupovými právy lze samozřejmě manipulovat běžnými příkazy jazyka SQL. Nicméně například tabulka ***user*** se časem značně rozrostla. Je celkem obtížné manipulovat s jejími záznamy standardními SQL příkazy. Proto MySQL disponuje speciálními příkazy pro manipulaci s uživatelskými účty a jejich privilegiemi. Například:

```
CREATE USER, DROP USER, RENAME USER, SET PASSWORD FOR atd.
```

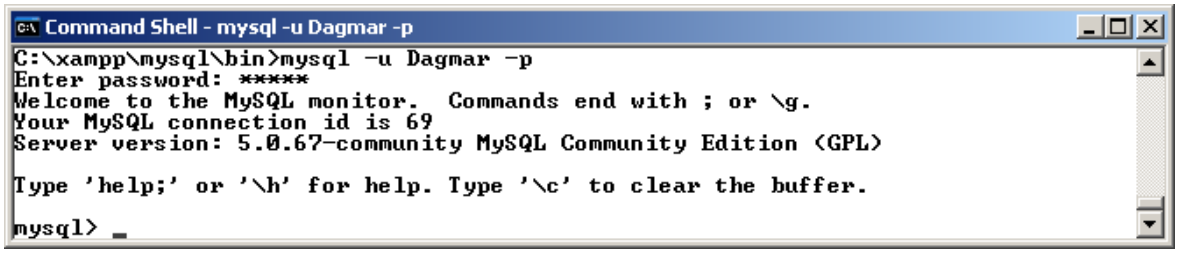
1.1.3 Autorizace

Autorizace je druhým krokem na cestě k datům. V první fázi přihlašování jsme byli serverem MySQL úspěšně identifikováni. Ve druhé fázi se ověřuje, co vlastně uvnitř systému můžeme a nemůžeme dělat. Je to stejné, jako když vstoupíme do panelového domu. Autentizace pro nás znamená, že máme klíče od hlavního vchodu. V domě je 32 bytů. Nyní nastane fáze autorizace. Do jakých bytů se můžeme dostat? Pouze do těch, ke kterým máme klíče. Typicky do svého. Uživatel *root* má všechna oprávnění. V podstatě jako by měl klíče od všech bytů. A nejenom to. On má klíče i od všech zásuvek v bytech.

Autentizace je tedy identifikace osoby.

Autorizace je tedy seznam oprávnění pro identifikovanou osobu.

Příklad je zcela jednoduchý. V předešlé kapitole jsme si založili uživatelku *Dagmar*. Zkusme se přihlásit pod jejím nově vytvořeným účtem (Obr. 23).



Obr. 23. Mysql - autentizace uživatele Dagmar.

Jsme přihlášení a nyní si vypišme seznam databází příkazem „SHOW DATABASES“. Systém vypíše pouze jedinou dostupnou databázi. Konkrétně databázi ***information_scheme***. Tato databáze je v MySQL od verze 5. Informuje o MySQL, její struktuře a běhu. Přístup k ní mají všichni. Je to systémová databáze. Modifikuje si ji pouze systém sám. Pro nás nyní nemá žádný význam. Ostatní databáze jsou pro uživatelku *Dagmar* nyní neviditelné. Jenom namátkou požádáme o vstup do databáze ***mysql*** příkazem „USE mysql“ (Obr. 24).

Obr. 24. Mysql - pokus o vstup do databáze mysql.

Pokus není úspěšný. Uživatelka *Dagmar*, z počítače „localhost“, nemá žádná oprávnění na databázi ***mysql***. Ukážeme si, jak taková oprávnění pro *Dagmar* získáme.

Zde, i v následujících kapitolách, již budeme pracovat s daty a databázemi. Proto si vytvoříme vlastní databázi a budeme manipulovat s ní. V bakalářské práci [8] jsou operace pro vytvoření databáze popsány pomoci PhpMyAdmin. My si vytvoříme databázi se jménem ***pokus***. Postupně se propracujeme k přístupovým právům pro uživatelku *Dagmar* k této nové databázi. Vytvoření provedeme v řádkovém klientu mysql. Přihlásíme se jako *root*, protože *Dagmar* nemá oprávnění k vytváření databází (Obr. 25).

Obr. 25. Mysql - založení nové databáze.

Databáze ***pokus*** je založena, ale *Dagmar* se do ní stejně nedostane. Uživatel *root* musí povolit *Dagmar* vstup do databáze ***pokus***. *Root* by také mohl povolit *Dagmar*, aby si databázi vytvořila sama. Tato alternativa je možná, ale předpokládá se, že *Dagmar* přesně ví, co dělá. V databázi ***mysql*** máme tabulku ***user***. O té již víme, že slouží pro autentizaci uživatelů. V této tabulce však můžeme navíc přidělovat globální oprávnění. Je to proto, že bezprostředně po úspěšné autentizaci je uživatel postaven dovnitř systému MySQL, ale jakoby před vrstvu databází. Tedy jako by stál na sídlišti.

Další fází je výběr databáze, se kterou hodlá pracovat. Nebo může manipulovat s databázemi jako takovými. Globální privilegia v tabulce ***user*** tedy stanovují, kdo a jak může manipulovat se samotnými databázemi a databázovým strojem MySQL. V této tabulce jsou tedy velice mocná privilegia a povolit neznalému uživateli jakékoliv zásahy na úrovni tabulky ***user,*** je čistý hazard. *Dagmar* by mohla bez mrknutí oka omylem smazat

databázi ***mysql***. Pokud by měla příslušná oprávnění. To by znamenalo fatální konec pro MySQL. Pravidlem tedy je, že uživatelům, o kterých víme, že nic neví, nepřidělujeme globální privilegia. *Dagmar* je pro nás naprosto normální uživatelka, který neví nic o fungování MySQL.

Nicméně i běžný uživatel v rámci nějakého klienta (internetový obchod či domácí účetnictví), musí mít možnost nějak manipulovat s daty ve své databázi. Povolíme *Dagmar* vytváření tabulek v databázi ***pokus***. Jenomže jak? V tabulce ***user*** to nesmíme dopustit. Podíváme se proto na jiné tabulky v databázi ***mysql***. Zajímat nás bude tabulka ***db***. Zde se ukládají přístupová práva nižší úrovně než v tabulce ***user***. Konkrétně přístupová práva v rámci specifikovaných databází. Podívejme se na výpis tabulky ***db*** (Obr. 26).

←T→	Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv
☐ ✎ ✕	localhost	phpmyadmin	pma	Y	Y	Y	Y	N

Obr. 26. PhpMyAdmin - výsek tabulky db.

Vidíme jeden záznam. Uživatel *pma* z počítače „localhost“ může v databázi ***phpmyadmin*** provádět příkazy SELECT, INSERT, UPDATE, DELETE. Jinak vůbec nikdo. Nebudeme zde vypisovat všechna privilegia. Ta jsou popsána v [1], [2], [3]. Důležité pro nás je, že uživatel *pma* nemohl v tabulce globálních privilegii ***user*** provádět téměř nic. S výjimkou vypnutí serveru. Jako *pma* uživatel nemůže MySQL poškodit. Může však zlobit s vypínáním serveru MySQL. To jediné má v tabulce ***user*** povoleno. V tabulce ***db*** má již ale mnoho oprávnění a může tedy provádět běžné příkazy. Ani zde však *pma* nemůže zakládat a mazat tabulky uvnitř databáze ***phpmyadmin***. Parametr *Create_priv* je nastaven na „N“.

Vraťme se nyní k *Dagmar* a povolme jí zakládání tabulek v databázi ***pokus***. Musíme do tabulky ***db*** přidat záznam o *Dagmar* (Obr. 26). Nastavíme jí všechna oprávnění mimo jediného. Nebude moci mazat tabulky.

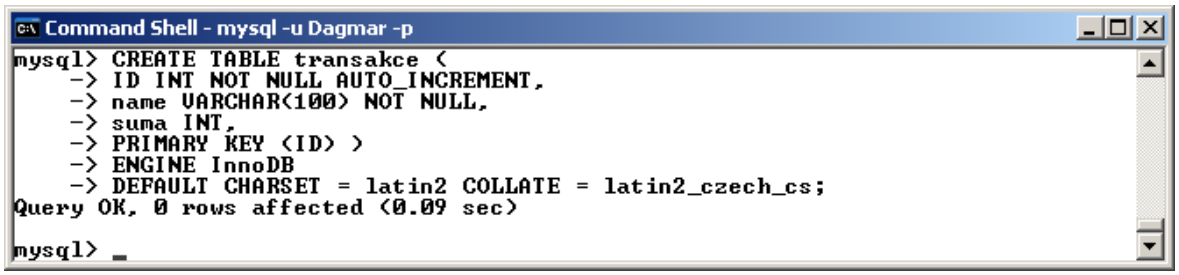
←T→	Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv
☐ ✎ ✕	localhost	phpmyadmin	pma	Y	Y	Y	Y	N	N
☐ ✎ ✕	localhost	pokus	Dagmar	Y	Y	Y	Y	Y	N

Obr. 26. PhpMyAdmin - oprávnění pro uživatele Dagmar v tabulce db.

Použijme klienta mysql a proveďme následující dva příkazy:

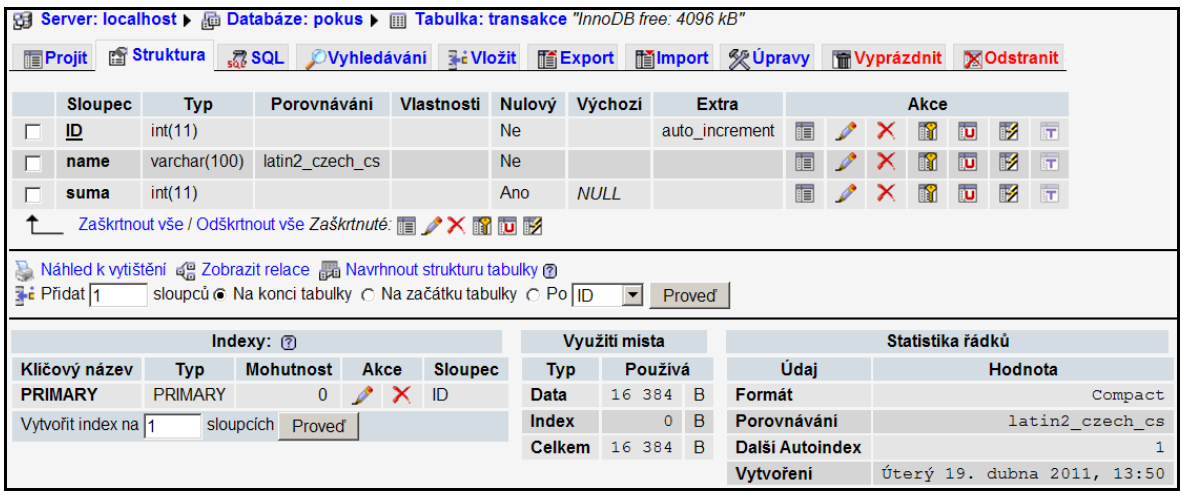
```
GRANT ALL ON pokus.* TO 'Dagmar'@'localhost';

REVOKE DROP ON pokus.* FROM 'Dagmar'@'localhost';
```

V prvním příkazu jsme nastavili uživatelce *Dagmar*, která se bude hlásit z počítače localhost, všechna privilegia. Ve druhém jsme odvolali privilegium pro mazání tabulek. Nyní může *Dagmar* zakládat do databáze ***pokus*** nové tabulky a přidávat do nich záznamy. Se záznamy bude moci plně manipulovat. Takže se přihlásíme jako *Dagmar* a v databázi ***pokus*** založíme tabulku se jménem ***transakce***. V této tabulce založíme 3 sloupečky. *ID, name a suma* (Obr. 27).



Obr. 27. Mysql - založení nové tabulky transakce.

Vypadá to složitě. Určitě je snadnější použít klienta PhpMyAdmin. Nicméně tabulku máme založenou. Vše proběhlo hladce (Obr. 28).



Obr. 28. PhpMyAdmin - struktura tabulky transakce.

Nyní se pokusí *Dagmar* o smazání tabulky. *Dagmar*, nikoliv PhpMyAdmin! *Dagmar* použije příkaz: „DROP TABLE transakce“ (Obr. 29).

Obr. 29. Mysql - pokus o smazání tabulky transakce.

Nemá nárok. Odebrali jsme jí oprávnění pro mazání tabulek. Možná proto, že od přírody všechno ráda maže a administrátor na to přišel.

Takto tedy funguje autorizace. Za zmínku ještě stojí několik poznámek k příkazům GRANT a REVOKE. Oba příkazy provádí nejenom změnu privilegii, ale také se postarají, aby vešly okamžitě v platnost. Nemusíme tedy používat příkaz „FLUSH PRIVILEGES“. Na samotné autorizaci se ještě podílí i další tabulky z databáze ***mysql***. Jedná se o stále nižší úroveň privilegií. Na závěr této kapitoly si shrneme autentizaci i autorizaci z pohledu MySQL kompletně.

1.1.4 Autentizace a autorizace – shrnutí

V databázi ***mysql*** je mnoho tabulek. Každá z nich určuje hladinu či hloubku přístupu k datům. Nejprve se použije tabulka ***user***. Při autentizaci, na základě tabulky ***user,*** jsou prověřovány v první fázi tyto údaje.

Tabulka ***user*** - autentizace.

1. Může se přistupující uživatel připojit k serveru MySQL? Odpovídá nějaký řádek tabulky ***user*** přihlašovacím údajům? Pokud ano, tak pokračujeme dalším krokem. Když ne, tak není spojení umožněno.
2. Nepřekročil přihlašovaný uživatel maximální počet přípustných spojení se serverem za hodinu? Pokud ne, tak pokračujeme krokem 3. Pokud ano, není spojení umožněno.

3. Požaduje přihlašovaný uživatel nějaký typ bezpečného spojení? Dodal k tomu potřebné dokumenty? Pokud ano, nastává krok 4. Pokud ne, je spojení ukončeno.

Tabulka ***user*** - autorizace.

4. Pořád pracujeme s tabulkou ***user***, ale nyní již probíhá autorizace. Prvním krokem je ověření počtu požadavků. Tedy, jestli přihlášený uživatel nepřekročil počet povolených požadavků za hodinu. Nebo nějaký jiný časový úsek. Pokud ano, je odhlášen. Pokud ne, nastává krok 5. Uživatel je připojen k serveru MySQL. Omezení v podobě počtů dotazů SQL je zavedeno z důvodu omezení konzumace prostředků. Je to nástroj, jak vyvážit vytížení serveru mezi větší počet uživatelů.
5. Má přihlášený uživatel nějaká globální oprávnění zapnuta (nastavena na Y)? Pokud ano, tak nemá cenu kontrolovat další tabulky. Oprávnění tabulky ***user*** mají prioritu. Takový uživatel může manipulovat s celými databázemi. Samozřejmě dle příslušných oprávnění. Pokud jsou oprávnění vypnuta (nastavena na N), nastává krok 6.

Tabulka ***db*** - autorizace.

6. Uživatel nemá v tabulce ***user*** žádná oprávnění. Musíme proto zkontrolovat záznamy v tabulce ***db***. V ní jsou oprávnění na konkrétní databázi zevnitř. Není tedy možné manipulovat s databázemi, ale pouze s tabulkami uvnitř konkrétní databáze. Prohledají se záznamy a opět se hledá totožný záznam. Pokud se najde, je autorizace skončena a uživatel může nakládat s daty uvnitř konkrétní databáze, na základě definovaných privilegií. Pokud není nalezen identický záznam, nebo je nalezen identický záznam bez uvedení položky „host“, přejdeme ke kroku 7.

Tabulka ***host*** – autorizace.

7. Tabulka ***host*** rozšiřuje možnosti tabulky ***db*** o rozlišovací schopnost na úrovni počítače, ze kterého se k databázi MySQL přihlašujeme. Je tedy možně nastavit rozdílná privilegia stejnému uživateli, který se hlásí z různých počítačů. Opět

se hledá záznam, který odpovídá přihlašovacím údajům. Když je záznam nalezen, je autorizace ukončena. Když ne, tak nastává krok 8.

Tabulka ***tables_prviv*** - autorizace.

8. MySQL jde s autorizací ještě dál. Pokud nenalezne záznamy v tabulkách ***user***, ***db*** ani ***host,*** tak se podívá do tabulky ***tables_priv***. V ní jsou oprávnění pouze pro manipulace s tabulkami nebo sloupci. Pokud je nalezen záznam, může takový uživatel manipulovat s tabulkami. Proces autorizace je ukončen. Pokud není záznam nalezen, pokračujeme krokem 9.

Tabulka ***columnes_priv*** - autorizace.

9. V poslední tabulce jsme na nejnižší možné úrovni. Na úrovní sloupečků uvnitř konkrétní tabulky. Když se nenajde shoda ani zde, tak to znamená definitivní konec. Takový uživatel se sice může k databázi přihlásit, ale to je vše. Nemůže naprosto nic. Je to stejné jako uživatelka *Dagmar*. Po jejím vytvoření neměla žádné oprávnění. K databázovému stroji MySQL se však dokázala připojit, protože byl nalezen záznam v tabulce ***user***. Poté jsme jí nastavili privilegia na databázi ***pokus***. Tam už pak mohla začít řádit. Zamezili jsme jí jenom mazání tabulek uvnitř databáze ***pokus***.

To je celý autentizační a autorizační proces databáze MySQL. Je velice dobře propracovaný. Spousta vytvořených databázových programu nevyužívá tak detailních přístupových práv. Obvykle přiřadí uživateli potřebné oprávnění na úrovni tabulky ***db***. Maximálně pak ***host***. O zbytek se stará klient sám. Je to taková pojistka proti příliš pracnému programování. Není ani tak problém zjistit, jaké má ten či onen uživatel oprávnění. Problém je zajistit, aby klient automaticky blokoval nedostupné položky ve svých formulářích. Popřípadě, aby je nezobrazoval vůbec. Pokud by totiž někdo provedl ruční zásah do databáze a změnil oprávnění, tak by se změnilo chování programu. Formuláře by mohly ztratit smysl. Porušila by se konzistence a integrita dat. Mohly by selhat databázové triggery a funkce atd. To je obrovské množství problémů. Jednodušší je nastavit oprávnění na nezbytně nutná. Program pak sám zabrání uživateli smazat záznam,

když k tomu nemá oprávnění. Jedná se sice o porušení bezpečnosti, ale přepokládáme, že uživatel nemá žádnou možnost se k datům dostat jinak, než přes databázový program.

Následující kapitola bude pojednávat o transakcích a jejich využití. Pokud má někdo zájem dozvědět se více o problematice privilegií databázového stroje MySQL, tak je ideálním pomocníkem referenční manuál přímo od výrobce [14]. Nesmí ho však zaskočit kapacita tohoto dokumentu. Má více jak 3000 stran. Na druhou stranu je tam snad všechno. Praktické ukázky přidělování oprávnění uživatelům nyní vynecháme. Aplikujeme je v transakcích.

1.2 Transakce

Transakce jsou některými programátory zavrhovány jako zbytečnost. Jiní naopak tvrdí, že se bez nich neobejdou. Transakce však existují. Je třeba je brát v potaz. Rovněž tak databáze MySQL transakce podporuje. Pouze však pro tabulky **InnoDB**. Proto se jimi budeme zabývat. Co to tedy ta transakce vlastně je?

Transakce je speciální postup manipulace s daty. Musíme v každé možné situaci udržet konzistenci dat. Navíc se snažíme o udržení atomicity kritických operací. To ještě není úplná definice transakce. Obecně musí transakce splňovat pravidla, pro která se vžilo označení **ACID** (Atomicity, Consistency, Isolation, Durability).

1. Atomicity – znamená, že se v rámci transakce buď provedou všechny příkazy, nebo ani jeden. Bez ohledu na zdroj problému.
2. Consistency – souvisí s pojmem referenční integrity. Pokud při transakci dojde k porušení referenční integrity, musí být transakce automaticky odvolána příkazem ROLLBACK. Zajistíme tím konzistenci dat.
3. Isolation – je převelice těžká věc. Každá transakce nesmí ovlivnit jinou transakci. Jinak řečeno, pokud jedna transakce manipuluje s daty ABC, tak s nimi v tu samou chvíli nesmí manipulovat jiná transakce. Jinak by došlo k souběhu nebo uváznutí.
4. Durability – je trvanlivost existence transakce. Princip transakce spočívá v tom, že se operace neprovádí s daty v tabulce přímo. Data, kterých se transakce týká, se pouze označí. To proto, aby jiná transakce dodržela izolaci. Fyzicky se změny provedou v separátním pomocném souboru. Po provedení ROLBACK se změny

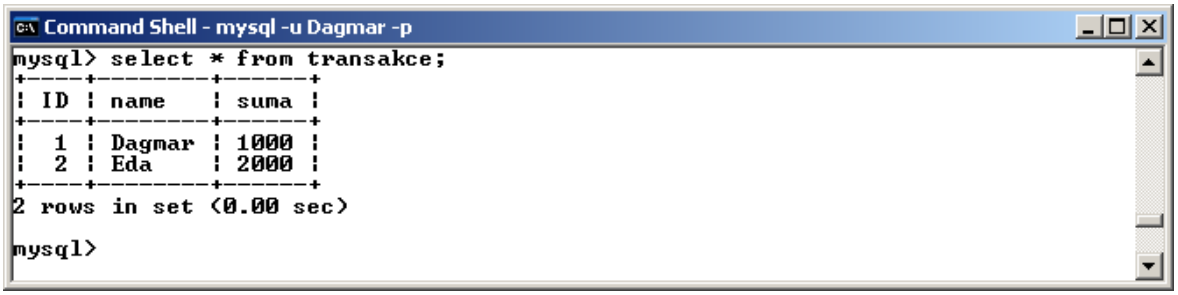
prostě zahodí. Po COMMIT se začnou zapisovat do skutečné tabulky. Až se tam zapíší všechna data a odstraní se zámky, pak teprve můžeme smazat pomocný soubor. Postup je o malinko složitější, ale pro představu dostačující. Trvanlivost je vlastně podržení si transakčních příkazů tak dlouho, jak je to potřeba.

1.2.1 Jednouživatelské transakce

Nejlepší budou ukázky, na kterých si vysvětlíme principy fungovaní transakcí. V předchozích kapitolách jsme si založili databázi ***pokus*** a v ní tabulku ***transakce***. Nyní nastal čas, abychom ji začali používat. Přihlásíme se do MySQL jako *Dagmar* a vstoupíme do databáze ***pokus***. Naplníme tabulku ***transakce*** nějakými daty. Stačí 2 záznamy:

```
INSERT INTO transakce VALUES (1,´Dagmar´,1000) ;
INSERT INTO transakce VALUES (2,´Eda´,2000) ;
```

Tabulka transakce bude mít dva záznamy (Obr. 30).

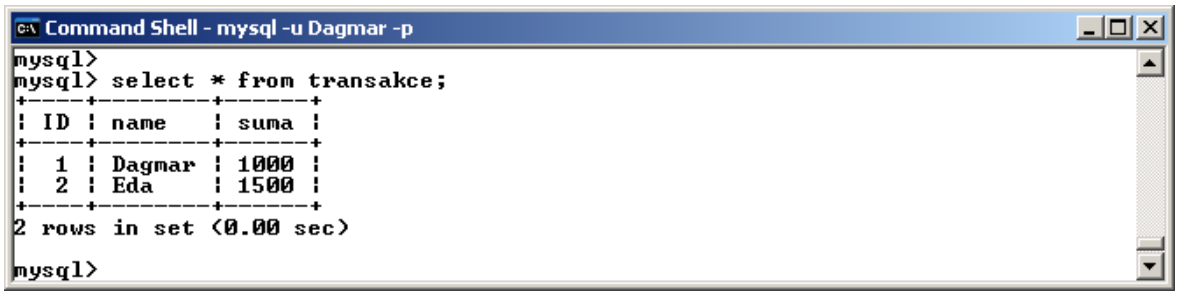


Obr. 30. Mysql - výpis tabulky transakce.

Nyní provedeme simulaci havárie serveru MySQL v průběhu provádění nějaké operace. Například výpadek napájení. Představme si, že *Eda* chce poslat *Dagmar* 500 korun. Klient (například banka) musí provést dvě operace. Nejprve odečte zákazníkovi *Eda* 500 Kč. V druhé operaci přičte *Dagmar* 500 Kč. (Pozorný čtenář se jistě zamyslí. A co poplatek 15,50 Kč? Berme to tak, že se nejedná o českou banku.) Takže to nyní provedeme. V mysql vykonáme první příkaz:

```
UPDATE transakce SET suma=suma-500 WHERE name='Eda';
```

Odečetli jsme zákazníkovi *Eda* 500 Kč. Druhý příkaz nestihneme provést, protože zhavaruje server MySQL. To lze nasimulovat snadno. Zavřeme okno s klientem mysql. Jak říkají programátoři – natvrdo. Znovu nastartujme mysql klienta a podívejme se na obsah tabulku transakce (Obr. 31).



Obr. 31. Mysql - tabulka transakce po havárii.

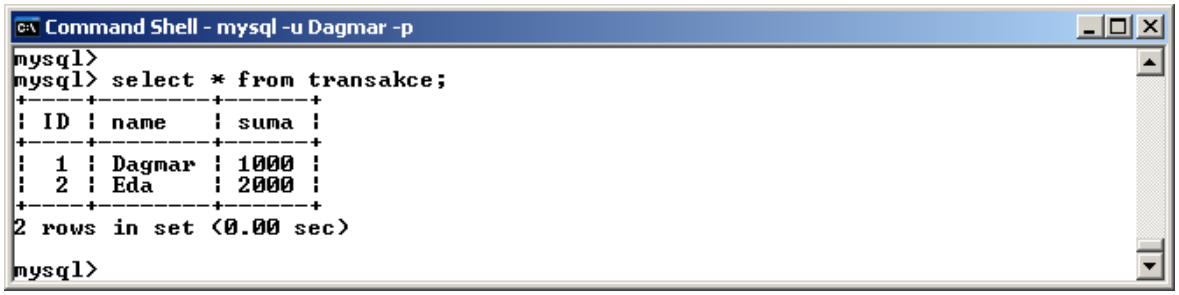
Mezitím volá Eda Dagmaře. „Ahoj Dášo, poslal jsem ti těch 500 Kč. Tak si je užij.“ „Mně žádných 500 nepřišlo“, povídá Dáša. „Jak nepřišlo, já to mám už dva dny odečtené z účtu“, zlobí se Eda. V bance se zřejmě stala chyba. Programátor nepoužil jednu maličkost. Nepostaral se o tzv. atomicitu operace. Jinými slovy nepoužil transakce. Zkusme si tu samou situaci zopakovat ještě jednou. Tentokrát transakce použijeme. Nejprve vraťme *Edovi* těch 500 Kč, aby měl zase svých 2 000 a Dagmar 1 000 Kč.

```
UPDATE transakce SET suma=suma+500 WHERE name='Eda';
```

Nyní bude sled příkazů před havárií trošku jiný. Použijeme na začátku SQL příkaz „START TRANSACTION“. Cely výpis příkazu bude následující:

```
START TRANSACTION;
UPDATE transakce SET suma=suma-1000 WHERE name='Eda';
```

Pak opět odstřelíme okno s klientem mysql. Druhý příkaz se tedy také neuskuteční. Po znovunastartování klienta se podíváme, co se stalo se záznamy v tabulce transakce. Vypadá to dobře (Obr. 32).

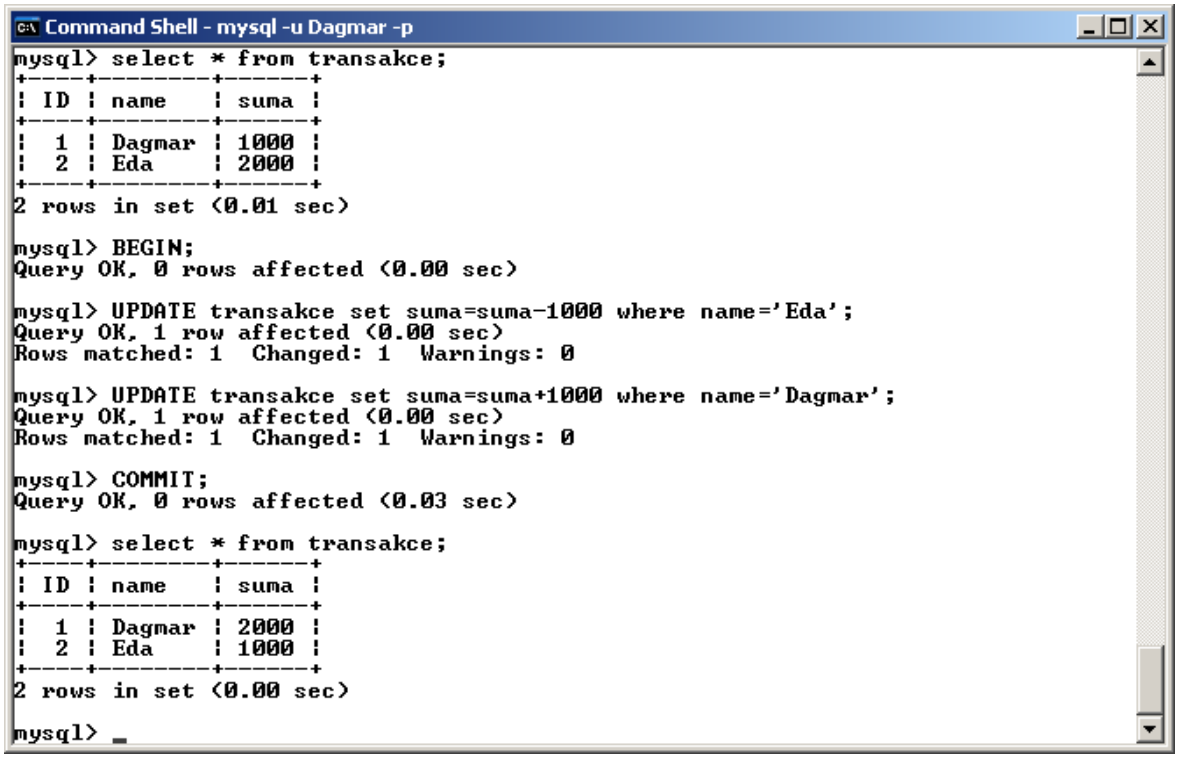


Obr. 32. Mysql – nedokončená transakce.

Databázové záznamy zůstaly nezměněny. A to i přesto, že první příkaz byl již proveden. Jak je to možné? Pokud převádíme peníze z účtu *Eda* na účet *Dagmar*, tak musíme provést dvě operace. Má to však jednu podmínku. Buď se provedou obě operace, nebo ani jedna. V našem případě jsme provedli pouze první a na druhou už nedošlo. Postarala se o to právě klauzule „START TRANSACTION“. Celá syntaxe transakčního zpracování má následující schéma.

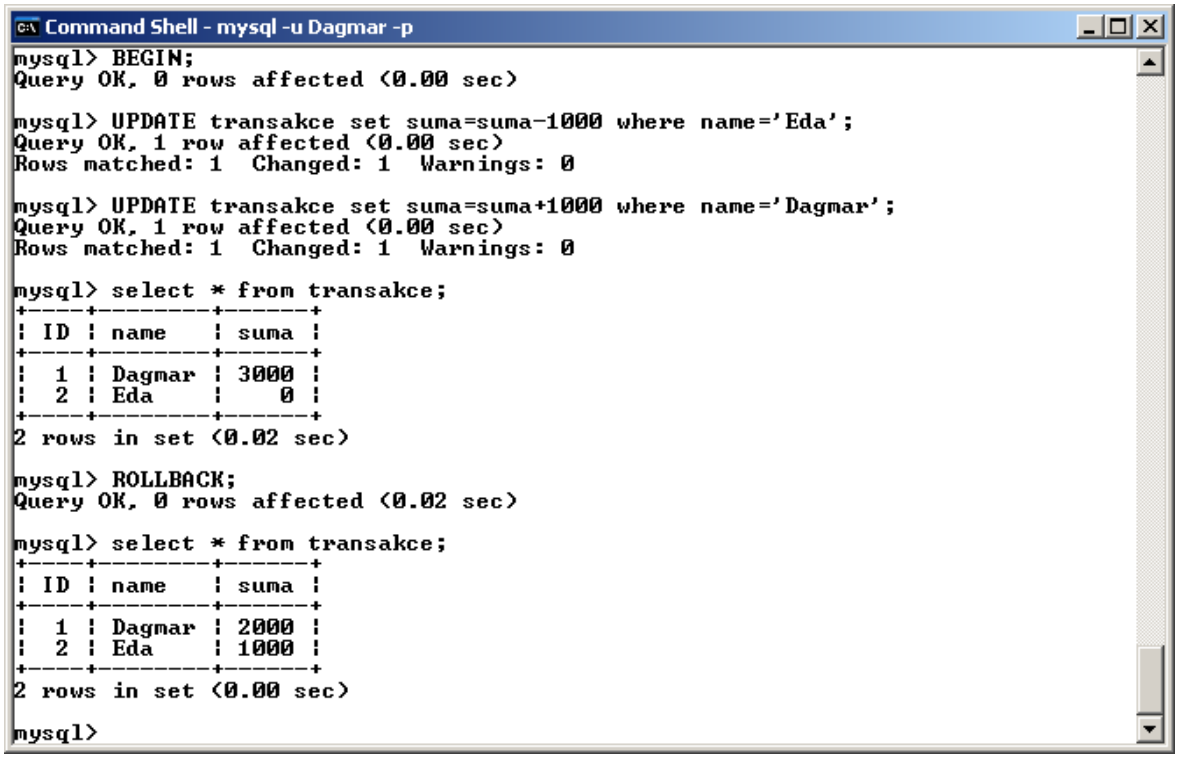
1. START TRANSACTION nebo BEGIN označují začátek transakce.
2. Sada příkazů SQL.
3. COMMIT nebo ROLLBACK transakci ukončují.

START TRANSACTION a BEGIN je to samé. Pouze označují začátek transakce a je lhostejné, který příkaz použijeme. ROLLBACK a COMMIT jsou dvě rozdílné věci. ROLLBACK ukončuje transakci s tím, že odvolává provedené operace. To znamená, že se po příkazu BEGIN nic nevykoná. Když tedy dojde k nějaké havárii, je to jako by se zavolal ROLLBACK. Nic se neprovede. Naopak příkaz COMMIT říká. Všechny operace po BEGIN proběhly bez problému. Můžeme je tedy potvrdit. Tomuto postupu se říká atomická operace. Nebo také nepřerušitelná operace. Opět si ukážeme příklad. Už nebudeme způsobovat havárii. Na následujícím obrázku (Obr. 33) bude sled operací, který zcela objasní fungování transakcí.



Obr. 33. Mysql - dokončená transakce příkazem COMMIT.

Provedli jsme dva příkazy UPDATE a potvrdili jsme je COMMIT. Transakce byla dokončena. Následoval výpis tabulky transakce. Vše proběhlo hladce. Ne tak ovšem na obrázku (Obr. 34). *Dagmar* chtěla uloupit z účtu *Eda* dalších 1000 Kč na nové boty. Byla však zvědavá a ještě před tím, než oba příkazy potvrdila příkazem COMMIT, tak se musela podívat, jestli má penízky na svém účtu. Vypadalo to, že tam jsou. Vtom však přišel domů *Eda*. Ihned poznal, která uhodila a než se *Dagmar* vzpamatovala, napsal příkaz ROLLBACK. A zachránil si svých 1000 Kč. Samozřejmě si ihned změnil heslo ke svému účtu. Na narozeniny pak koupil *Dagmar* papuče z bazaru za 12,50 Kč.



Obr. 34. Mysql - odvolaná transakce příkazem ROLLBACK.

To je všechno ohledně transakcí v jednouživatelském režimu. Je třeba říci ještě jednu věc. V MySQL se u tabulek InnoDB standardně považuje každá operace za transakci. Pokud napíšeme sled operací UPDATE, tak se každý jednotlivý příklad považuje za dokončenou transakci. Například:

```
UPDATE transakce SET suma=suma-1000 WHERE name='Eda';
```

Ve skutečnosti se na pozadí provede sled následujících příkazů.

```
START TRANSACTION;
UPDATE transakce SET suma=suma-1000 WHERE name='Eda';
COMMIT;
```

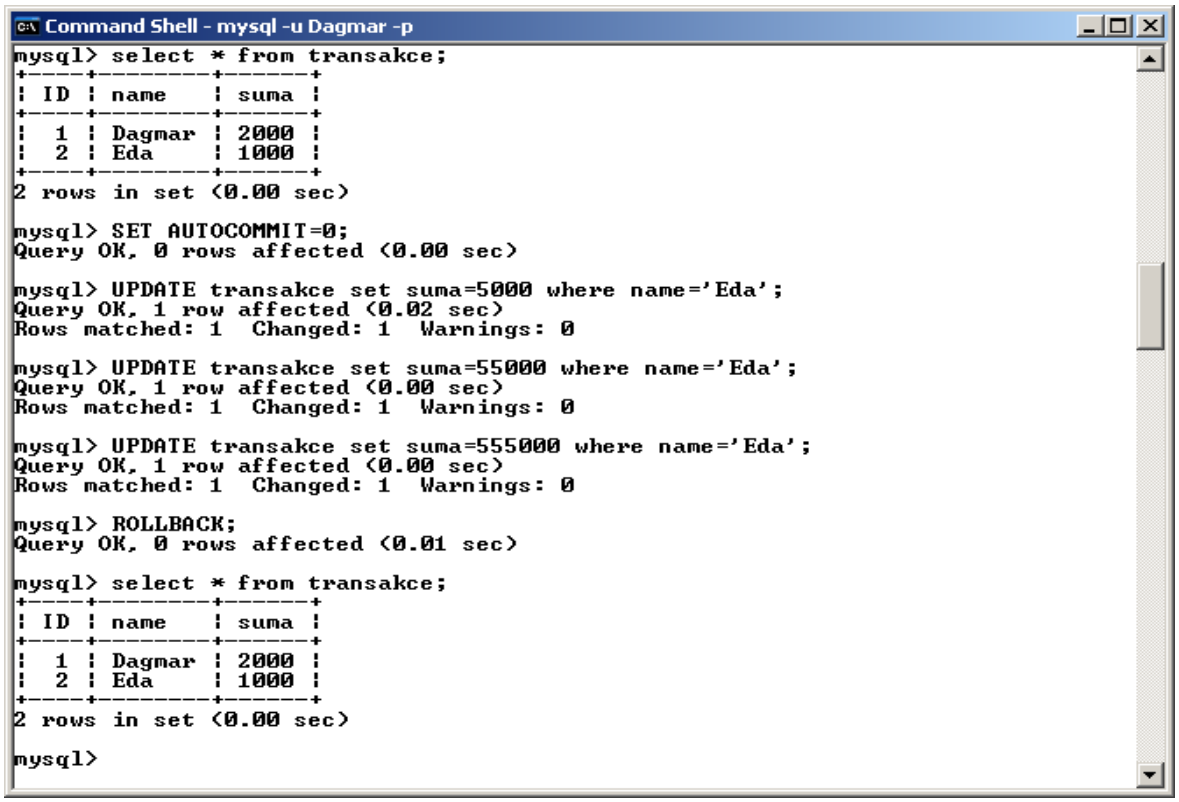
Na svědomí to má konfigurační direktiva AUTOCOMMIT. Pokud je nastavena na 1, tak jsou všechny operace s databází považovány za tzv. mikrotransakce. Pokud přepínač nastavíme na 0, tak přepneme na supertransakce. To znamená, že příkazy SQL budou považovány za jedinou transakci až do použití příkazu ROLLBACK nebo COMMIT. O tom jak je přepínač nastaven, se lze snadno dozvědět pomocí příkazu SELECT:

```
SELECT @@autocommit ;
```

Nastavení přepínače pak vyřešíme příkazem:

```
SET AUTOCOMMIT = 0 ; SET AUTOCOMMIT = 1 ;
```

Na následujícím obrázku si ukážeme variantu supertransakce. (Obr. 35).



Obr. 35. Mysql - supertransakce.

Nyní je již zcela jasné, co způsobí přepínač AUTOCOMMIT. Kdybychom ho nechali nastavený na 1, tak by se záznamy v tabulce ***transakce*** měnily s každým příkazem UPDATE. Takto však byly příkazy izolovány až do použití příkazu ROLLBACK, nebo COMMIT. Na obrázku jsme použili ROLLBACK. To znamená, že všechny předchozí příkazy byly anulovány. AUTOCOMMIT nastavený na 0 je celkem nebezpečný. Pokud na něho zapomeneme, tak bez použití ukončovacího příkazu pro transakci, můžeme vytvořit gigantickou transakci. Pokud nakonec dojde k nějaké chybě, tak přijdeme o množství dat. Je to stejné, jako bychom zapomněli ukládat soubor ve Wordu a on potom celý zhavaroval. V následující kapitole se budeme zabývat transakcemi z trošku jiného pohledu.

1.2.2 Transakce ve víceuživatelském prostředí

Až doteď byla práce s transakcemi a jejich pochopení zcela triviální záležitost. Pokud ovšem s databází pracuje mnoho uživatelů současně, je situace diametrálně odlišná. Vzhledem k tomu, že transakce musí splňovat pravidla ACID, tak nastávají nutně problémy. Zejména s dodržením izolace transakcí. Máme-li v tabulce 100 záznamů, mohou transakce pracovat současně. Ovšem pouze za předpokladu, že každá transakce bude ovlivňovat jiné záznamy. V tomto případě není žádný problém. Může a běžně nastává situace, kdy dvě transakce mění tytéž záznamy. Nebo alespoň část záznamů je společných. V takovém případě přece nejsou transakce izolovány. A to je nepřípustné. Izolace je nutná. Proto ovladač InnoDB jednu transakci zablokuje. Počká se na dokončení první transakce. Teprve potom je povoleno pokračovat druhé transakci. Pokud první transakce trvá příliš dlouho, tak je druhá transakce stornována ROLLBACK. Tento postup je jediný bezpečný. Bohužel záleží právě na rychlosti zpracování první transakce. Ono pokud je první transakce omylem supertransakce a úroveň izolace nastavena na „serializable“, pak se nedostane na žádnou další transakci. V totální izolaci se totiž transakce řadí do fronty a vykonávají se postupně. Následné transakce by byly jedna po druhé odvolávány.

Úrovní izolací je víc. V podstatě se v nich balancuje bezpečnost s výkonem. Totální izolace je nejlepší, ale nejpomalejší. O izolacích si řekneme v další kapitole. Izolace jako takové nejsou jediným problémem, se kterým se musíme vypořádat. Co se stane, když transakce mění data, která si v tu chvíli čte někdo jiný? Například měníme záznam na řádku 123, kde chceme přepsat položku sumy z 25 000 na 20 000 Kč. V tu samou chvíli se však jiný uživatel na tuto položku dívá a provádí s ní matematické výpočty. Transakce dávno změnila sumu, ale ještě ji nepotvrdila COMMIT. V databázi jsou proto staré údaje. Příkaz SELECT také reaguje na izolace a zámky, které tam transakce vkládají. SELECT si může počkat na dokončená data. Nebo také nemusí. Záleží to na mnoha kombinačních nastaveních. Tím problémy nekončí. Stejně tak jako v případě resetu uživatelských práv, musíme ještě řešit klientské vyrovnávací paměti. Klient často mění nebo prohlíží data uložená ve své interní paměti. Změny pak může posílat do databáze najednou. Například bude chtít změnit záznam, který už v databázi mezitím někdo vymazal. Tento postup vyvolá výjimku. Pokud není ošetřena, způsobí pád aplikace. I klient tedy musí počítat se všemi problémy, které víceuživatelský režim vytváří.

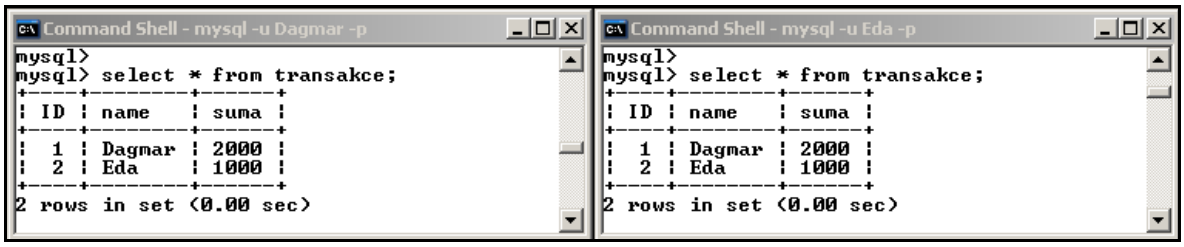
Byly uvedeny základní problémy víceuživatelského zpracování dat. Není jich vůbec málo. Představme si, že v databázi jsou desetitisíce záznamů. Ty jsou navíc rozděleny do separátních tabulek a pospojované přes klíče. Řekněme 50 tabulek. Na takovouto databázi se přihlásí 1 000 uživatelů v jediný okamžik. Je zcela jisté, že dojde ke křížení požadavků. Nejenom v rámci jediné tabulky, ale křížem přes tabulky. To si v podstatě ani nelze představit. Pouhé zachování referenční integrity, při použití transakcí, musí být pro ovladač InnoDB nesmírně těžké.

Nechme však teorie. Ta je tak složitá, že s přehledem vystačí na samostatnou diplomovou práci. Věnujme se raději příkladům. Pokusme se nasimulovat situace, ve kterých budou viditelné alespoň základní problémy s transakcemi a zamykáním. Některé se nám však nemusí podařit nasimulovat. Rychlost databáze je značná. Simulace nedokumentovaného souběhu nebo uváznutí bude pravděpodobně nerealizovatelná. Nicméně základní problémy lze hezky ukázat. Ovladač InnoDB je výborně odladěný. I on se však může dostat do situace, kdy souběhu a uváznutí nedokáže zabránit. Není tedy zcela stoprocentní.

1.2.3 Transakce - příklad 1

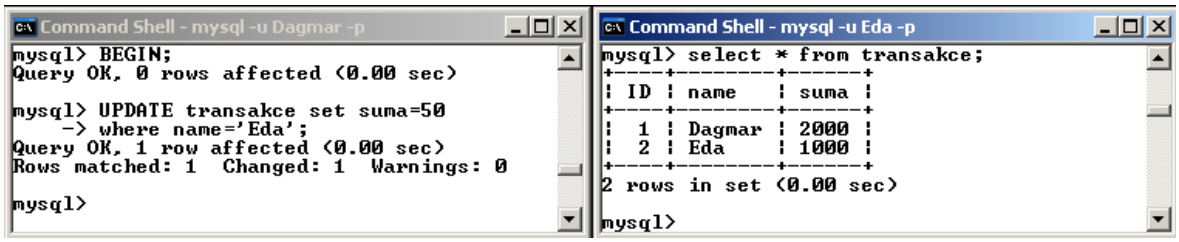
Začneme běžnou transakcí. Použijeme opět tabulku ***transakce***. S uživatelkou *Dagmar* už máme zkušenosti, tak ji využijeme. Hesla k jiným účtům jí však neprozradíme.

Mějme v databázi přihlášené dva uživatele. *Dagmar* a *Edu*. *Dagmar* bude měnit záznamy a *Eda* se na ně bude v tu samou chvíli dívat. Uvidíme, jaká data budou k dispozici. Budeme potřebovat dva klienty mysql. Je to jednoduché. Nastartujeme dvě okna. Do jednoho přihlásíme *Dagmar* a do druhého se přihlásí uživatel *Eda*. U obou nastavíme pracovní databázi ***pokus***. U *Edy* vyvoláme chybu, protože jsme mu nenastavili přístupová práva na databázi ***pokus***. Takže svou chybu napravíme. Už víme jak. Použijme třeba PhpMyAdmin. Poté spustíme v obou oknech příkaz SELECT (Obr. 36).



Obr. 35. Mysql - připojení dvou uživatelů na tabulku transakce.

Nyní *Dagmar* spustí transakci. Pak vykoná modifikaci nějakého záznamu. Třeba záznamu „name=Eda“, ve kterém modifikuje položku suma na 50. Ještě před tím, než transakci ukončí příkazem COMMIT, se *Eda* (v pravém okně) podívá příkazem SELECT na výpis tabulky transakce (Obr. 37).

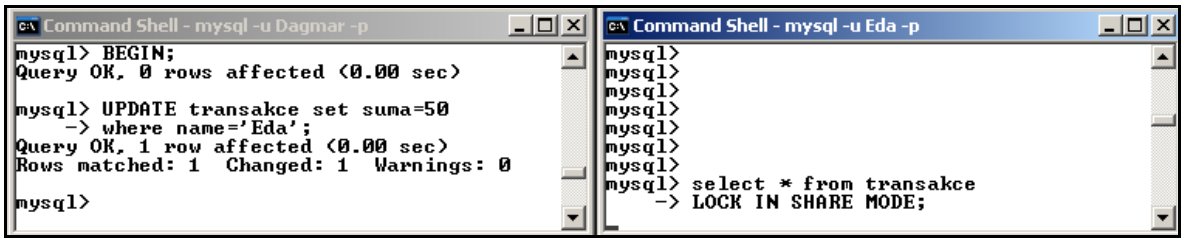


Obr. 37. Mysql - SELECT za běhu transakce.

Eda získal původní data. Tabulka ***transakce*** ještě neví, jak se *Dagmar* rozhodne, nebo jestli nedojde k chybě.

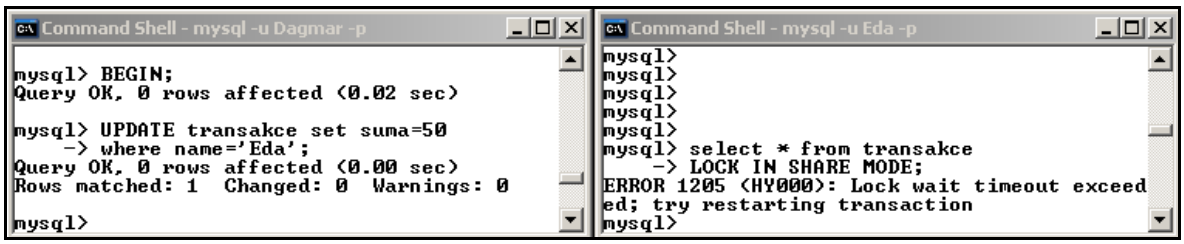
1.2.4 Transakce - příklad 2

Ve druhém příkladu si ukážeme, jak pozastavit příkaz SELECT do doby, než se transakce ukončí. Postupovat budeme obdobně. *Dagmar* zůstává stejná. Transakce stále běží. Není zatím ukončená ani ROLLBACK ani COMMIT. V okně *Edy* požádáme o výpis SELECT s dodatkem LOCK IN SHARE MODE (Obr. 38).



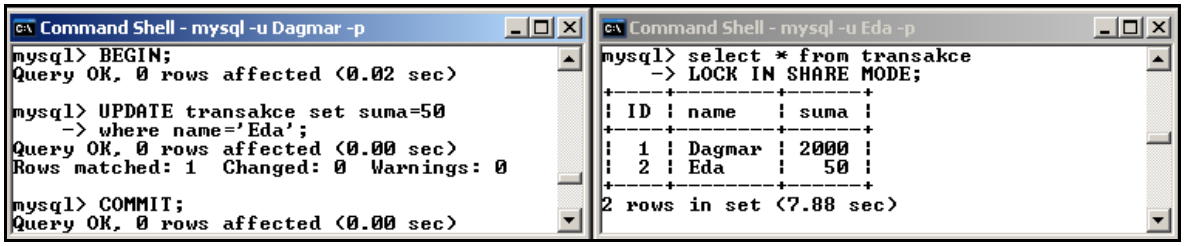
Obr. 38. Mysql - blokováni dotazu SELECT.

Proces bude blokován. Nyní nastanou dvě možné situace. Pokud bude transakce *Dagmar* trvat déle než 50 sekund, tak SELECT *Edy* bude stornován ovladačem (Obr. 39).



Obr. 39. Mysql - timeout pro dotaz SELECT.

Pokud však transakce *Dagmar* skončí dříve, než zmíněný časový interval, tak *Eda* dostane svůj výsledek. Buď modifikovaná data, nebo původní data. Záleží na tom, jestli *Dagmar* ukončí transakci COMMIT nebo ROLLBACK (Obr. 40).



Obr. 40. Mysql - úspěšné vykonání transakce i dotazu SELECT.

Dagmar ukončila transakci příkazem COMMIT. *Eda* tedy dostane již modifikovaná data. Existuje ještě příkaz SELECT …… FOR UPDATE. Jeho chování je podobné, jako LOCK IN SHARE MODE. Tyto příkazy se dají použít i uvnitř transakce. Jejich chování je závislé na stupni izolace transakcí. Než si ukážeme další příklad na transakce, je dobré vědět něco málo o těchto izolacích [2].

1.2.5 Izolace transakcí

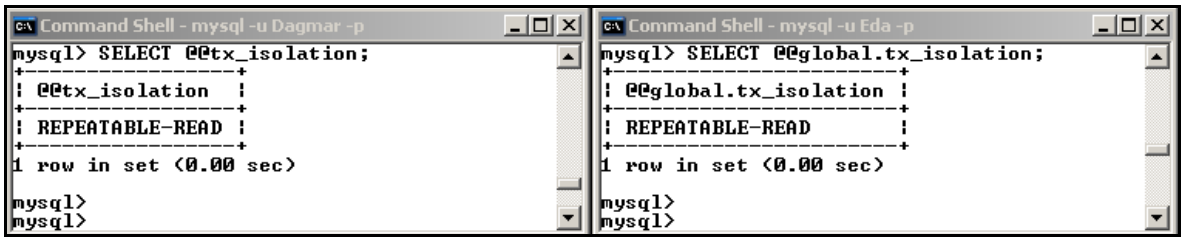
Ovladač InnoDB je standardně nastaven na jistou úroveň izolace. Je na ni optimalizován. Existují 4 úrovně. Můžeme je měnit u každé transakce. Syntaxe je následující.

SET *rozsah_použití* TRANSACTION ISOLATION LEVEL *typ_použití*

Rozsah_použití definuje, kdy a na co se nastavení vztahuje. Pokud *rozsah_použití* vynecháme tak bude *typ_použití* platit pouze pro aktuální transakci. Pokud je použito SESSION, tak změna platí pro všechny transakce až do další změny nebo do konce spojení. Nakonec GLOBAL nastaví izolace pro všechna nová spojení s databází. Neplatí však v tom spojení, kde se to nastavilo. *Typ_použití* má 4 možnosti.

1. READ UNCOMMITED – tady se SELECT chová tak, jako by žádné transakce nebyly. Vidíme zmodifikovaná data ještě před tím, než je transakce potvrzena COMMIT nebo odvolána ROLLBACK. SELECT tedy nepodléhá izolaci. Nicméně ostatní příkazy jako INSERT nebo UPDATE již ano. Někdy se to může hodit.
2. READ COMMITED – zde se SELECT chová trošku izolovaněji. Zobrazí změny, ale pouze pokud transakce skončila COMMIT. Nicméně jeden a ten samý SELECT může vracet různé výsledky v rámci jedné transakce. To proto, že se v databázi neustále mění data. Těch COMMIT ukončení může být 100 za vteřinu.
3. REPEATABLE READ – tady je SELECT zcela izolován. Nemůže dojít k tomu, aby v rámci jedné transakce vracel různé výsledky. Je to jako bychom zmrazili ostatní data a transakce.
4. SERALIZABLE – totální izolace. SELECT se používá s LOCK IN SHARE MODE. Je to vlastně, jako bychom každou transakci vykonávali postupně. Jednu za druhou. De facto se pro každou transakci zamkne rovnou celá tabulka a je po problémech.

InnoDB je defaultně nastaven na izolaci REPEAPABLE READ. Aktuální stav zjistíme příkazem `SELECT @@global.tx_isolation` nebo `SELECT @@tx_isolation` (Obr. 41).



Obr. 41. Mysql - izolace klientů.

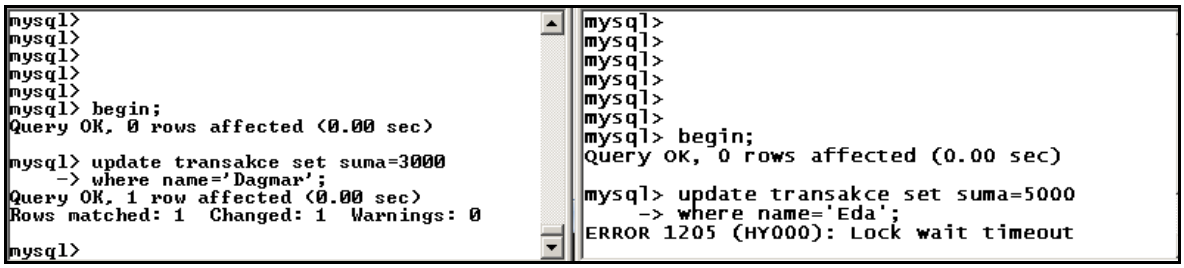
Příklady na chování příkazu SELECT při různých stupních izolace si zde již nebudeme uvádět. Každý si je může nasimulovat sám. Postup je stejný jako u příkladu 1 a 2. My si probereme jiné situace s transakcemi.

1.2.6 Křížení transakcí

V následujícím příkladu se podíváme na situaci, kdy *Eda* i *Dagmar* budou měnit stejný záznam. Nejprve *Dagmar* spustí transakci a provede modifikaci jednoho ze záznamů. Transakci nebude ukončovat. Potom *Eda* provede běžný příkaz UPDATE na ten samý záznam, co *Dagmar* v transakci. Výsledkem bude blokování příkazu UPDATE *Edy* až do chvíle, než *Dagmar* dokončí svoji transakci, nebo bude překročen timeout. V případě překročení timeoutu, se příkaz *Edy* stornuje. Nesmíme zapomenout, že UPDATE *Edy*, nebyl proveden jako transakce. To však není podstatné. Přepínač AUTOCOMMIT je nastaven na 1. To znamená, že i když UPDATE *Edy* nebyl vložen po návěští BEGIN, tak se stejně na pozadí tento příkaz sám aktivoval. Jednalo se o mikrotransakci. Z této skutečnosti vyplývá, že každý, kdo používá databázový engine InnoDB a má defaultní nastavení, používá automaticky transakce. I když pouze mikrotransakce.

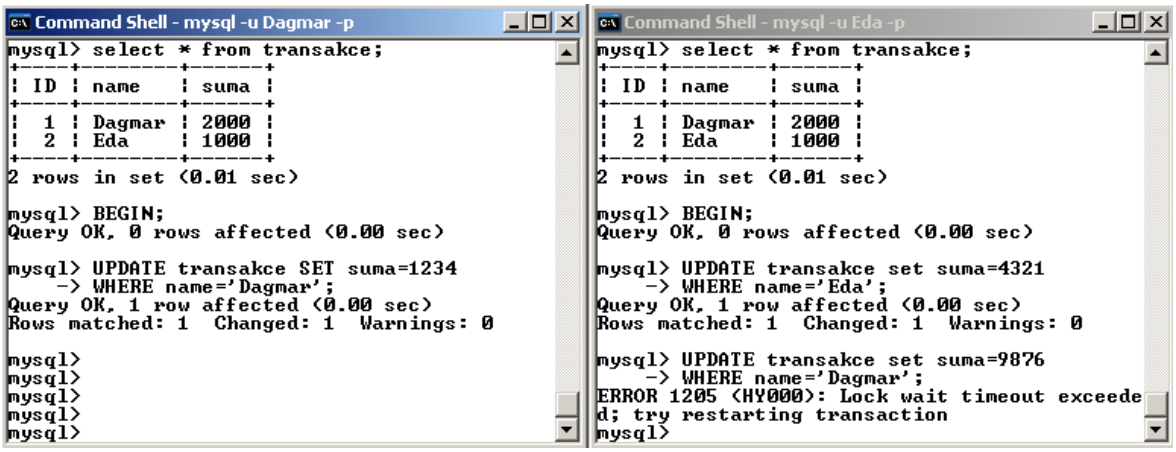
1.2.7 Křížení transakcí – příklad 1

Tento příklad je velice důležitý. Při vytváření tabulky ***transakce*** jsme se dopustili velmi zásadní chyby. Nepoužili jsme na tabulku ***transakce*** žádný index. To má dalekosáhlé důsledky na chování transakcí. Mějme opět naše záznamy v tabulce ***transakce***. *Dagmar* spustí transakci a modifikuje sumu u záznamu se svým jménem. Transakci zatím neukončí. *Eda* potom také spustí transakci a pokusí se o modifikování sumy zase u svého jména. Nastane nepochopitelná situace. Transakce Edy je blokována (Obr. 42).



Obr. 42. Mysql - nestandardní chování transakce.

Jak je možné, že je *Eda* blokovaný? Transakce přece blokují na úrovni záznamů a ne celých tabulek. To pak nemusíme používat tabulky typu InnoDB, ale klidně typ **MyISAM**. Tam pak použijeme LOCK TABLES a výsledek bude stejný. Proč se trápit s transakcemi. Vysvětlení je neuvěřitelně prosté. V naší tabulce ***transakce*** nejsou žádné indexy. Transakce nemá podle čeho řadit zámky. Nedokáže izolovat konkrétní záznam. Nemůže vědět, jestli se mezi modifikované záznamy z jedné transakce nenapasuje záznam z jiné transakce. Byť zcela jiný záznam. Tak jednoduše zamkne kde co. V našem případě vlastně celou tabulku. Pokud nepoužijeme indexy, tak se v tabulce, která má 1 000 000 záznamů, nastaví třeba 5 689 zámků. To je celkem tragédie pro výkon databáze. Na druhou stranu je to krásně bezpečné. Takže abychom mohli efektivně využívat transakce, tak musíme vhodně nastavit indexy. V naší tabulce ***transakce*** nastavíme index na sloupec *name*. Použijeme k tomu klienta PhpMyAdmin. Následující příklad již pak bude v pořádku. Blokování bude probíhat pouze na stejných záznamech. Pokud budeme křížem modifikovat jiné záznamy, tak nenastane žádný problém. Na následujícím obrázku uvidíme, jak povolený UPDATE, tak blokovaný UPDATE (Obr. 43).



Obr. 43. Mysql - funkční transakce s blokováním na úrovni záznamu.

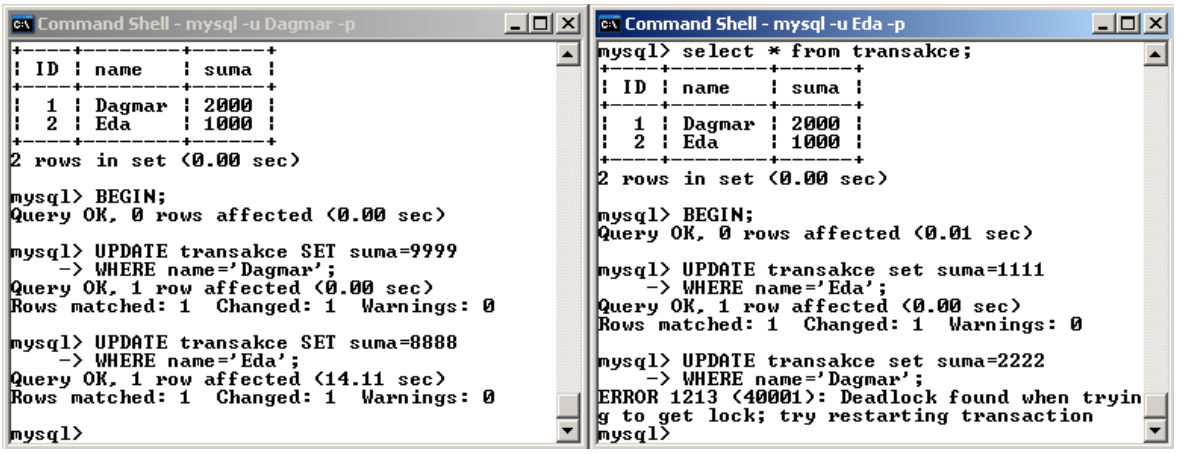
Je krásně vidět, že nyní už vše funguje jak má. Pokud *Dagmar* modifikovala svůj záznam v tabulce ***transakce,*** tak *Eda* mohl v rámci své transakce modifikovat zase ten svůj. Transakce *Dagmar* blokovala pouze záznam se jménem name=Dagmar. Potom se *Eda* ještě pokusil zmodifikovat záznam name=Dagmar a byl již odmítnut. Tento záznam byl uzamčen transakcí uživatelky *Dagmar*. Na obrázku mu pak vypršela expirace.

1.2.8 Křížení transakcí – příklad 2

V posledním příkladě si nasimulujeme uváznutí. Neboli deadlock. Mnoho programátorů se domnívá, že souběh a uváznutí není prakticky možné vytvořit. Databáze přece nesmí dopustit, aby k němu došlo. Tento pohled na věc není správný. K uváznutí i souběhu se může každá databáze dopracovat velice snadno. Důležité je, že je databázový ovladač dokáže identifikovat a zotavit se z nich. To znamená, že nezůstane viset nekonečně dlouho v této pasti. Nekompromisní přístup pak je, nečekat ani vteřinu a nemilosrdně zlikvidovat proces, který uváznutí způsobil. Zkusme si takový deadlock nasimulovat. Je to prosté. Nechme *Dagmar* i *Edu* spustit transakce. Příkazy provádíme postupně. Jednou *Dagmar,* pak *Eda,* potom zase *Dagmar* a nakonec *Eda*.

1. *Dagmar* si zmodifikuje záznam se svým jménem.
2. *Eda* si zmodifikuje záznam se svým jménem.
3. *Dagmar* se pokusí modifikovat záznam se jménem Eda. To není možné. Tento záznam je uzamčen v transakci *Edy*. Proto je *Dagmar* zablokována.
4. *Eda* se pokusí modifikovat záznam se jménem Dagmar. To také není možné. I *Dagmar* má ve své transakci tento záznam uzamčen.

Nastal čistý deadlock. Ani jeden nemůže pokračovat. Blokují se navzájem. InnoDB to zjistí a násilně ukončí transakci, která deadlock způsobila. Což je v pořádku. Celý postup je na obrázku (Obr. 44).



Obr. 44. Mysql - ukázka uváznutí (deadlock).

Uváznutí způsobil *Eda*. Jako poslední totiž do kříže modifikoval *Dagmar*. Přičemž sama *Dagmar* byla již blokována, protože se pokoušela modifikovat *Edu*.

1.2.9 Transakce – shrnutí

Problematika transakcí je velice složitá. Snad proto někteří programátoři odmítají transakce používat. Faktem zůstává, že je to velice užitečná věc. Zejména v oblastech s kritickými daty. Například bankovní účty. Je na každém, aby důkladně zvážil, jestli transakce potřebuje. Existují aplikace, kde transakce není nutné používat. Například aplikace, které slouží většinou pro čtení. S transakcemi souvisí celá řada činností, které musí programátor vědět. Jinak nebude v používání úspěšný. Třeba ony nešťastné indexy. Koho by to napadlo? Na druhou stranu bychom si bez podivného chování transakcí ani nevšimli, že jsme na indexy zapomněli. Výkon takové databáze by byl celkem žalostný. Transakce by vytvářela tisíce zbytečných zámků. A také vyhledávání bez indexů by nebylo nic moc extra rychlého. Inu všechno souvisí se vším.

1.3 Zabezpečení

V této kapitole zohledníme potřebu bezpečnosti ze strany klienta, databáze a spojení mezi nimi. V některých případech není zabezpečení nutné vůbec. V jiných je naopak kritické. Vždy se musíme podívat na účel programu. Zohlednit všechny bezpečnostní rizika a minimalizovat jejich účinky. Je několik pohledů na bezpečnost. Tedy, co máme chránit. Máme chránit klienta, vlastní databázi, či komunikaci klienta s databází. Aspektů

je samozřejmě víc. Například si můžeme položit otázku, jestli má cenu chránit klienta či databázi, když se postaráme o špičkové šifrování dat. V této situaci nám může být lhostejné, zda nám někdo zcizí data, databázi, či klientský program. Získané informace budou nečitelné a znalost či neznalost struktury databáze či klienta pro nás nebude podstatná. Jiný příklad zase posuzuje aspekt bezpečnosti a rychlosti systému. Pokud budeme provoz šifrovat tak snížíme výkon databáze. Šifrování bude probíhat jak na straně klienta, tak na straně serveru.

MySQL dokáže navazovat zabezpečená spojení. Navíc umí rozlišit, ze kterého PC se klient hlásí. Není proto problém přidělit jednomu uživateli jeden šifrovaný a druhý nešifrovaný kanál. Nešifrovaný použijeme v rámci firemního intranetu a šifrovaný zase zvenku. Tedy ze sítě Internet. Možností je opravdu celá řada. Ucelenou odpověď na úroveň zabezpečení lze obecně shrnout. Tato věta zde již byla jednou vyslovena. Jsou data, která chceme uložit do databáze pro nás nějak důležitá a osobní? Pokud ano, je namístě uvažovat o zabezpečení dat. Podívejme se na několik aspektů ohledně zabezpečení systému.

1.3.1 Bezpečnost serveru

Nyní se podíváme na vlastní data uložená v nějaké databázi na serveru. Zapomeňme, že existuje nějaký klient. Budeme se zabývat pouze počítačem, na kterém databáze běží. Jak ochráníme data před potencionálním útočníkem?

V prvé řadě musíme zabezpečit operační systém serveru. Fyzicky se nesmí neautorizovaná osoba do serveru dostat. Totéž se týká síťového provozu. Musíme se postarat, aby souborový systém s našimi daty byl bezpečný. To proto, že databázové tabulky jsou běžné soubory. Uzavřeme všechny porty, služby a programy, které nejsou pro běh serveru nutné. Nastavíme firewall, řádná přístupová práva pro vstup do počítače a samozřejmě také k daným souborům. Zejména administrátorská hesla by měla být dostatečně silná. V žádném případě by neměla být ukládána v nešifrované podobě. Snahou většiny útočníků je získat superuživatelský přístup. Pak mohou se serverem dělat cokoliv. Pokud jsme nuceni poskytovat na server přístupy jiného charakteru, tak se snažíme o minimální možné rozsahy přístupových práv. Další možností, jak ochránit data v tabulkách, je jejich šifrování. Tato alternativa může ochránit data i v případě zcizení databázových souborů. Platí však předpoklad, že útočník nezcizí i šifrovací klíče. Ty by

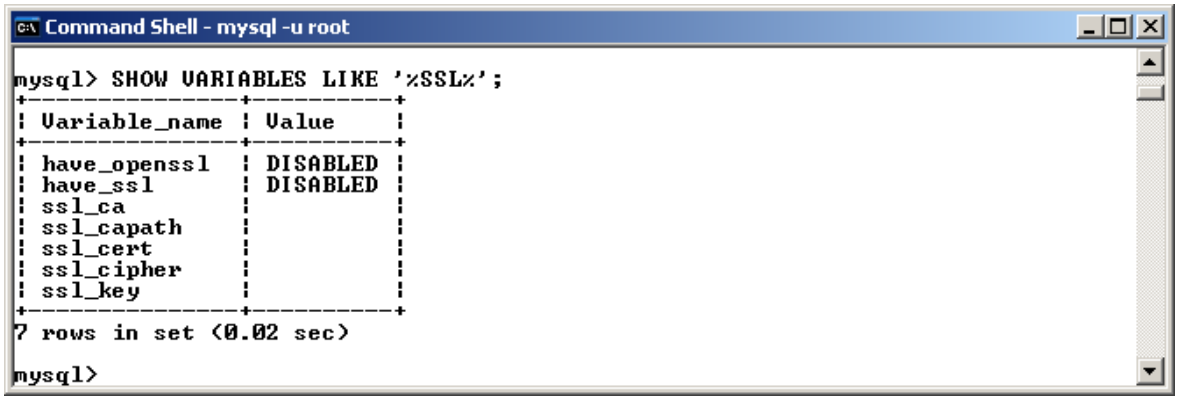
měly být kvalitně zabezpečeny a použity pouze pro start démona. Démon mysqld sám může být rizikem. Je to pouze program a ten může být napaden nebo mít chybu na úrovni kódu. Proto je nutné ho pravidelně aktualizovat. Minimálně při zjištění dokumentovatelné chyby. Měly by se sledovat informace o aktualizacích. Dalším pravidlem je nespouštět démona mysqld jako *administrátor*. Vytvoříme mu separátní účet s přístupy v rámci adresáře, kde je databáze umístěna. Démon pak nebude mít žádná práva k ostatním souborům operačního systému serveru.

1.3.2 Bezpečnost spojení

Jedná se o nejzranitelnější část. Servery a klienty máme fyzicky k dispozici. Vše mezi tím už ne. Spojení klienta se serverem můžeme zabezpečit šifrováním přenosového kanálu. Odposlech kanálu je pak sice stále možný, ale získaná data jsou nečitelná. MySQL disponuje podporou SSL. Pokud se chce klient spojit s MySQL šifrovaně, tak musí provést několik nutných úkonů.

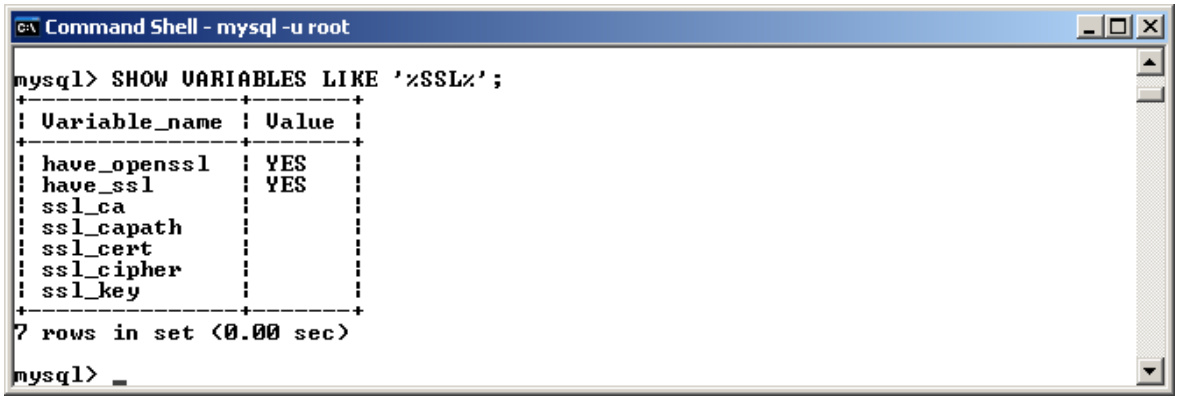
1. Server MySQL musí být spuštěn s podporou SSL a musí vůbec SSL podporovat.
2. Uživatel musí mít v oprávnění definovaný požadavek na SSL. Popřípadě další zpřesňující informace.
3. Klient se musí přihlašovat k serveru s nastavením potřebným pro SSL komunikaci.
4. Pro SSL komunikaci potřebujeme příslušné certifikáty

Stručně si popíšeme SSL v rámci MySQL. Detailně se tím zabývá [2], [5], [14]. Nejprve si ověříme, jestli naše verze MySQL vůbec SSL podporuje (Obr. 45).



Obr. 45. Mysql - dotaz na SSL funkcionalitu.

Náš server podporuje SSL. Je však spuštěn bez SSL. Nápravu sjednáme zastavením a následným spuštěním démona mysqld s přepínačem –ssl. Poté si opět ověříme, jestli již server bude schopen SSL komunikace (Obr. 46).



Obr. 46. Mysql - povolena komunikace SSL.

Nyní je server připraven na komunikaci prostřednictvím SSL. V tomto stavu však nebude nic fungovat. Server nemá žádné klíče ani certifikáty. Tyto certifikáty získáme ze strany nějaké certifikační autority. Samozřejmě pro osobní potřeby si můžeme certifikát vytvořit i sami. Například prostřednictvím programu OpenSSL [9], [10]. Získáme příslušné certifikáty a klíče jak pro server, tak pro klienta. Někde si je uložíme a s jejich pomocí nastartujeme démona mysqld.

1.3.3 OpenSSL – vytvoření certifikátu

XAMPP [10], který máme nainstalovaný, již podporuje OpenSSL. To znamená, že počítá i s protokolem SSL. Pokusme se nyní o vygenerování svých certifikátů. Přejděme do adresáře c:\xampp\apache\bin. Zde spusťme příkazový řádek. Nejprve si do systémových proměnných operačního systému zavedeme novou položku.

```
SET OPENSSL_CONF=C:\xampp\apache\bin\openssl.cnf
```

Jedná se o konfigurace OpenSSL. V konfiguračním souboru nebudeme provádět žádné nastavování. Nemá to nyní cenu. V ostrém provozu to samozřejmě cenu má. Při generování se nás program OpenSSL ptá na několik údajů. Z cvičných důvodů si je všechny vyplníme. Postup generování je následující.

1. Vytvoříme si vlastní CA. Vlastní proto, abychom nemuseli platit za vystavení serverového a klientského certifikátu u profesionální CA. Například WeriSign.

   ```
   openssl genrsa 2048 > ca-key.pem

   openssl req -new -x509 -nodes -days 1000 \
   -key ca-key.pem > ca-cert.pem
   ```

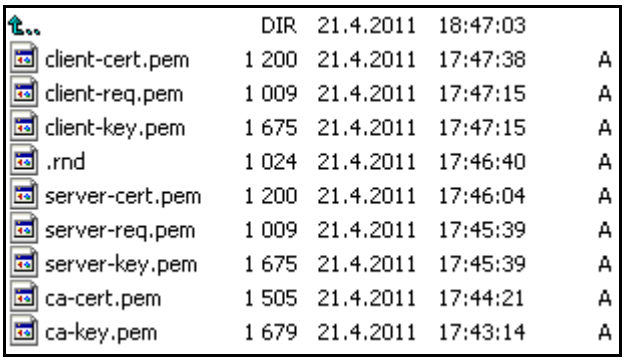
2. Serverový certifikát si vytvoříme tak, že vystavíme žádost o tento certifikát u nějaké certifikační autority. Buď za peníze u profesionální CA, nebo v tomto případě u naší vlastní CA. Poté vygenerujeme serverový certifikát typu x509 podepsaný naší CA. Na základě předchozí žádosti.

   ```
   openssl req -newkey rsa:2048 -days 1000 \
   -nodes -keyout server-key.pem > server-req.pem

   openssl x509 -req -in server-req.pem -days 1000 \
   -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem
   ```

3. Klientský certifikát se vystavuje stejně jako serverový.

   ```
   openssl req -newkey rsa:2048 -days 1000 \
   -nodes -keyout client-key.pem > client-req.pem

   openssl x509 -req -in client-req.pem -days 1000 \
   -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > client-cert.pem
   ```

Vygenerované soubory si přesuneme do nějakého adresáře. Třeba c:\test. Jedná se o tyto soubory (Obr. 47). Samozřejmě si je můžeme libovolně pojmenovat.



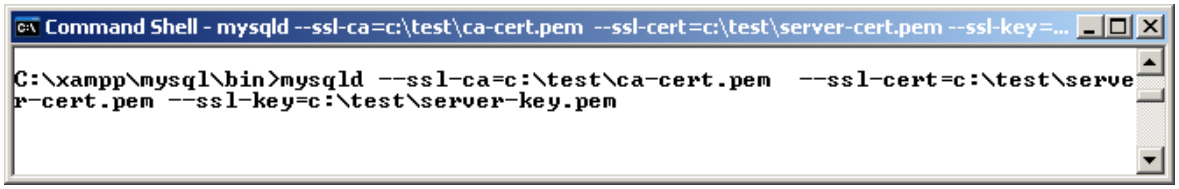
..	DIR	21.4.2011	18:47:03	
client-cert.pem	1 200	21.4.2011	17:47:38	A
client-req.pem	1 009	21.4.2011	17:47:15	A
client-key.pem	1 675	21.4.2011	17:47:15	A
.rnd	1 024	21.4.2011	17:46:40	A
server-cert.pem	1 200	21.4.2011	17:46:04	A
server-req.pem	1 009	21.4.2011	17:45:39	A
server-key.pem	1 675	21.4.2011	17:45:39	A
ca-cert.pem	1 505	21.4.2011	17:44:21	A
ca-key.pem	1 679	21.4.2011	17:43:14	A

Obr. 47. OpenSSL - certifikáty a klíče.

Nyní můžeme přistoupit k zabezpečení spojení. Více se o OpenSSL lze dočíst například zde [9]. V databázi MySQL si vytvoříme uživatele *Jan*. Potom tomuto uživateli nadefinujeme práva a zároveň SSL požadavek. To znamená, že uživatel *Jan* se bude muset prokázat nějakým certifikátem. Učiníme tak následujícím příkazem:

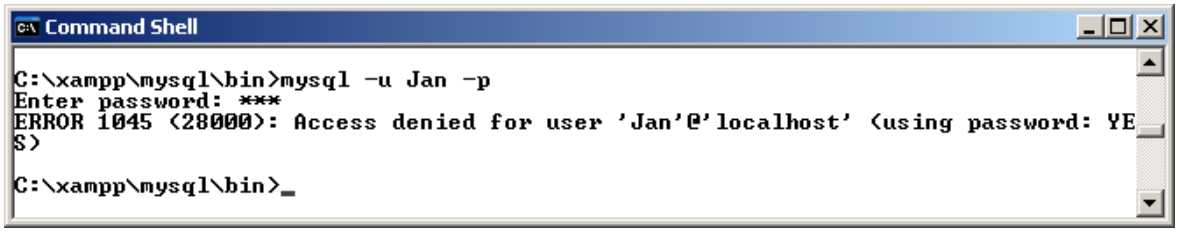
```
GRANT SELECT, UPDATE, INSERT ON pokus.* TO Jan@'' IDENTIFIED BY 'Jan'
REQUIRE SSL;
```

Právě REQUIRE určuje stupeň zabezpečení. My jsme zvolili nejjednodušší možnost. Server a klient si spolu pouze vymění klíče a šifrují. Server nezajímá, kdo *Jan* je. Proto se stačí k serveru připojit pouze s pomocí ca-cert.pem. Vyšší stupně pak musí použít i osobní identifikaci, identifikaci vůči třetí straně, typ šifry atd. Tak daleko zacházet již nebudeme. Uživatel *Jan* je založen a má nastaveno, že musí použít SSL. Nyní zastavíme MySQL server. Musíme démona spustit ručně s volbami SSL. Najdeme ho ve stejném adresáři, jako klienta mysql (Obr. 48). Použijeme vygenerované certifikáty a klíče.



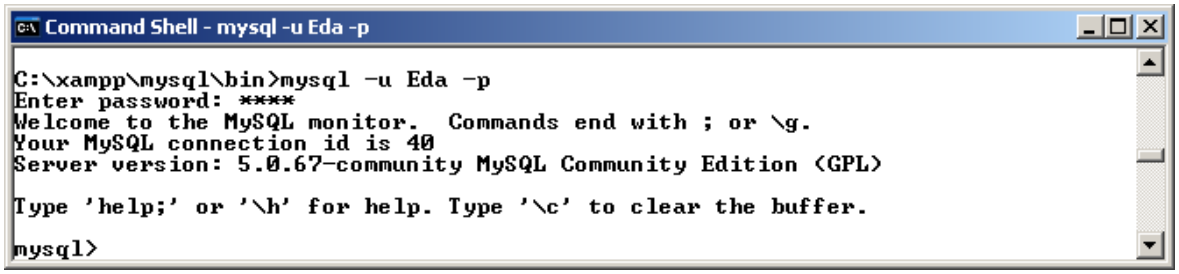
Obr. 48. Mysql - Démon mysqld s podporou ssl.

MySQL tedy běží a nám nezbývá než se připojit. Připojíme se jako *Jan*. Nejprve zcela běžným způsobem (Obr. 49).



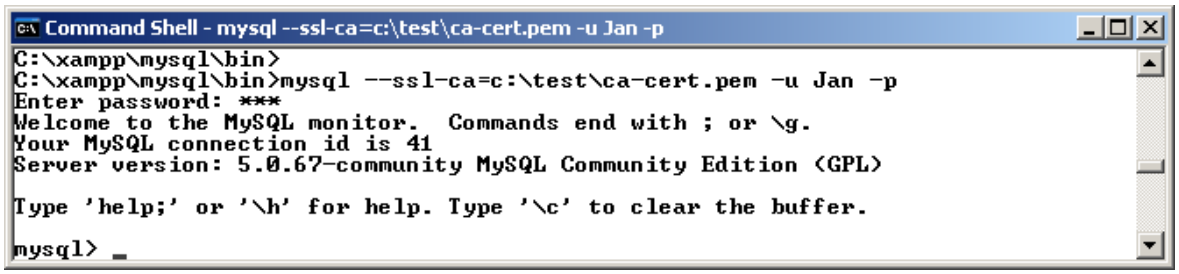
Obr. 49. Mysql - přihlášení uživatele Jan bez SSL.

Přihlášení skončí chybou přístupu. Server vyžaduje po uživateli *Jan* SSL. Jiný uživatel, který nemá v autentizaci požadavek na SSL, se může přihlásit zcela běžně. Třeba uživatel *Eda* (Obr. 50).



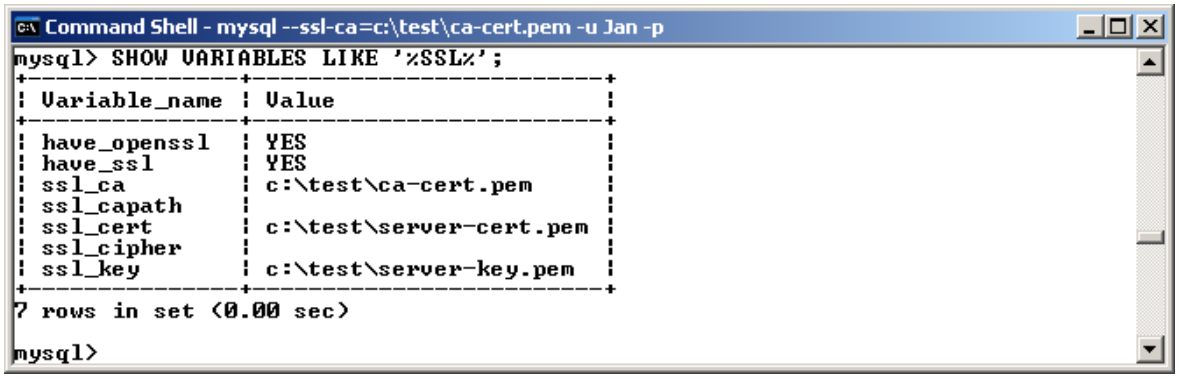
Obr. 50. Mysql - uživatel Eda bez SSL.

Musíme tedy uživateli *Jan* nějak pomoct a při přihlašování říct mysql, že chceme bezpečné spojení SSL (Obr. 51).



Obr. 51. Mysql - funkční SSL spojení nejnižší úrovně.

Záleží tedy skutečně na tom, jak je autentizován uživatel. *Jan* musí z počítače „localhost“ používat SSL. Mohli bychom mu povolit, že z jiného počítače by nemusel použit SSL. V tomto je MySQL hodně propracované. Ještě se podívejme na výpis SSL proměnných na běžícím serveru MySQL. Necháme *Jana* provést výpis, když už je přihlášený (Obr.52).



Obr. 52. Mysql - výpis SSL proměnných serveru MySQL.

`--ssl-ca=`*`ca-cert.pem`* Certifikační autorita. Podpis může být i náš vlastní.

`--ssl-cert=`*`server-cert.pem`* Veřejný klíč serveru.

`--ssl-key=`*`server-key.pem`* Privátní klíč serveru.

Ostatní nevyplněné položky dále specifikují SSL komunikaci. Například ssl_cipher definuje povolené šifrovací algoritmy a ssl_capath zase adresář, kde jsou certifikáty uložené.

```
--ssl-cipher='EDH-RSA-DES-CBC3-SHA';
```

My jsme vše provedli ručně. Pokud bychom požadovali po *Janovi* vyšší stupeň ověření, museli bychom mu nadefinovat k REQUIRE SLL ještě REQUIRE X509 plus další položky, jako REQUIRE ISSUER, SUBJECT, CIPHER. V podstatě by se *Jan* musel prokázat, že je to skutečně on, nebo dokonce by to musela potvrdit nezávislá certifikační autorita. Musel by použít typ certifikátu x509 a konkrétní šifrovací algoritmus. Jinak by ho server neautentizoval. *Jan* by se pak musel přihlásit například takto:

```
mysql --ssl-ca=c:\test\ca-cert.pem --ssl-cert=c:\test\client-cert.pem
--ssl-key=c:\test\client-key.pem
```

Do dalších podrobností již zabíhat nebudeme. Každý, kdo má zájem, si může další možnosti natrénovat sám. Tolik tedy k zabezpečení přenosového kanálu. Je to jenom jedna z možností. Použít bychom mohli třeba **VPN** tunel mezi klientem a serverem. V podstatě je to úplně stejné, jenom se o zabezpečení postará technologie VPN.

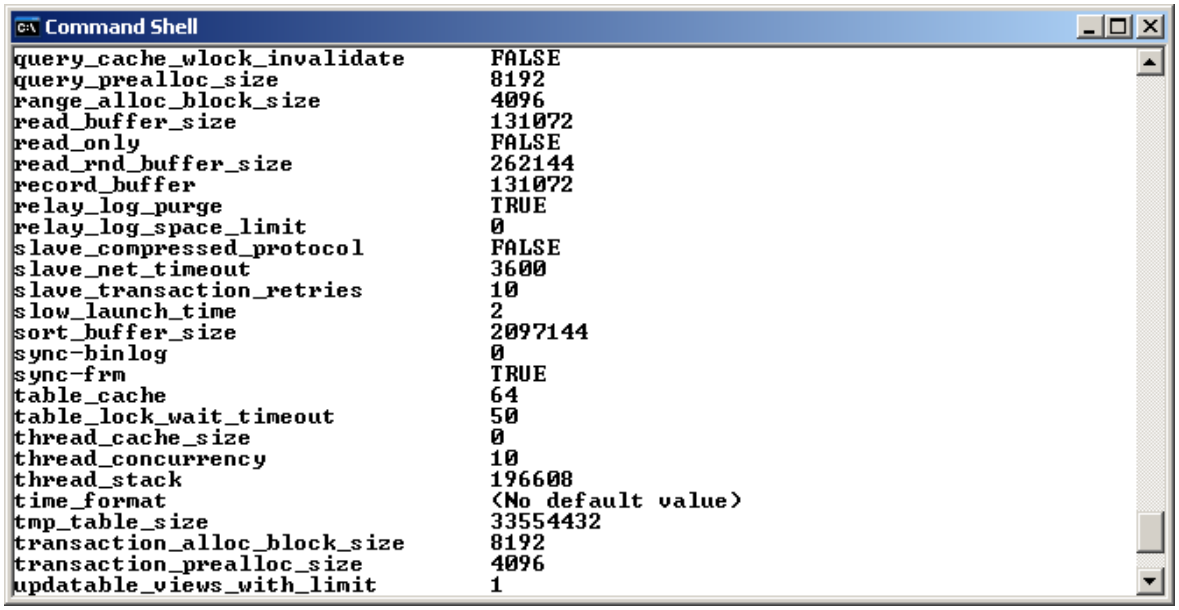
1.3.4 Bezpečnost klienta

Na závěr nám zbývá ještě bezpečnost klienta. Ta je benevolentní. Klient může být klidně zcela veřejný. Stačí si hlídat klíče. Nicméně je dobré vědět, zda je klient autentický. Je možné klienta hacknout a využít obecného stupně přístupových práv na serveru. Bezpečnost je pak zcela na straně serveru. Modifikovaný klient by mohl měnit SQL dotazy. Ideální je si klienta před každým použitím stáhnout z ověřeného webu. Nebo ho nosit sebou společně s certifikáty. Samozřejmě v zašifrované podobě. Často se pak používá webový klient postavený na PHP. Potom stačí použít webový prohlížeč, navázat bezpečné spojení a nechat server aby generoval stránky PHP. Tedy za předpokladu, že tyto stránky taky někdo nenaboura.

1.4 Navyšování výkonu mysql

Zpočátku života databáze příliš nemusíme řešit její výkon. Postupem času se velikost databáze zvětšuje a můžeme narážet na **HW** limity počítače, na kterém databáze běží. Pokud bude naše aplikace úspěšná a bude nasazena na internetu, může počet dotazů na databázi raketově růst. Pokud včas nezachytíme problémy s odezvou databáze na dotazy zákazníků, může to poškodit dobré jméno naší aplikace. To vše jenom proto, že databáze běží na pomalém HW nebo je špatně navržena. Optimalizaci můžeme provést na úrovni software nebo hardware. Softwarové optimalizace se provádí přímo s konfiguračními hodnotami databázového stroje MySQL. Je jich celá řada. Vždy je potřebné nastudovat potřebný materiál k dané verzi databáze. Nebudeme je zde uvádět, neboť je jich víc než 200. Vypsat si je můžeme takto (Obr. 53):

```
mysqld --verbose --help;
```



Obr. 53. Mysqld - fragment konfiguračních direktiv MySQL.

Další možností je pak klient mysql a příkaz „SHOW VARIABLES“. Dostaneme téměř totéž. Server MySQL bývá nastaven výborně. Kdo jiný by měl také vědět, jak na to, když ne přímo výrobce. Pro zajímavost si pár parametrů uvedeme.

- `key_buf_size` Jedná se o alokování paměti na indexy. Tím se odlehčí disku, protože je velká pravděpodobnost, že je příslušný index uložen v paměti RAM. Proces pak nemusí číst z disku.
- `max_connection` Počet maximálních simultánních připojení. Například hodnota 100 říká, že s databází bude moci pracovat najednou maximálně 100 klientů.
- `skip-networking` Přepínač dokáže zablokovat komunikaci **TCP/IP**. To se používá, když je aplikace i databáze na stejném PC.
- `long_query_time` Parametr udává maximální čas pro vykonání jednoho dotazu SQL.
- `log_show_queries` Tento parametr určí soubor, do kterého se budou zapisovat dotazy, které trvaly moc dlouho. Respektive překročily čas definovaný parametrem `long_query_time`. To je velice užitečné při ladění výkonu. Každý takový dotaz je z podstaty

nevhodný, neboť zpomaluje odezvy serveru. Je možné ho analyzovat a optimalizovat.

Parametru je opravdu hodně. Jejich vhodnou kombinací lze optimalizovat výkon MySQL. Přímo výrobce této databáze nabízí šablony, ve kterých jsou parametry nastaveny na různé typy hardware. Vycházejí z konfiguračního souboru „my.ini (cnf)“. Stejně tak i my můžeme zkopírovat tento soubor a upravit si ho podle své libosti. Změnami parametrů v konfiguračním souboru ovlivňujeme chovaní databázového stroje MySQL. Především jeho výkon. Je velice těžké ladit databázi, která není zatížena. Naopak není vhodné ladit databázi za plného běhu v ostrém provozu. Tedy ve chvíli, kdy teprve můžeme zhodnotit chovaní serveru. Výhodou ostrého ladění však je alespoň možnost monitorování provozu. Na základě monitorování zjistíme problémy zatížené databáze. Upravené konfigurační soubory a následný restart serveru MySQL pak můžeme realizovat v okamžiku minimálního provozu. Nastavování parametrů serveru MySQL je velice složitá věc. Závisí nejenom na hardware, ale také na struktuře databáze. Celá tato diplomová práce by nestačila na popsání všech parametrů a jejich činností. Zaměřme se spíš na hardwarové navyšování výkonu. To je věc, kterou výrobce MySQL nemůže ovlivnit. Pouze snad doporučit.

1.4.1 HW navýšení výkonu

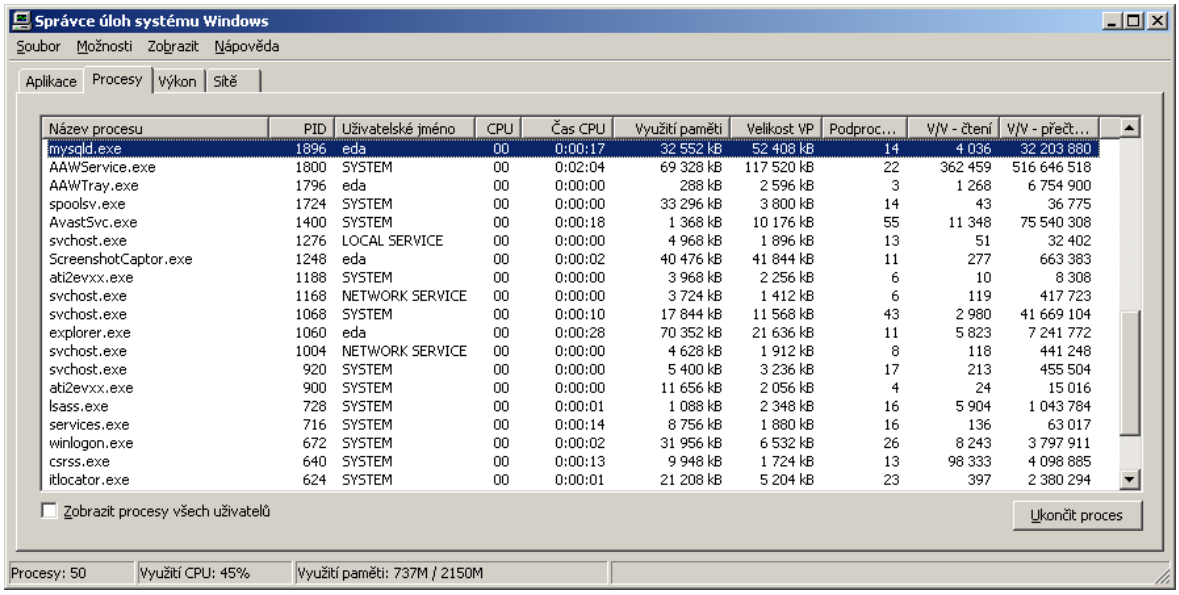
Databáze nám běží na současném HW. Doba odezvy na položený dotaz SQL se stále prodlužuje, až začne být neúnosná. Hrozí odliv zákazníků. Nezbývá nám nic jiného, než navýšit výkon hardware. V podstatě máme dvě možnosti.

1. Navýšit výkon stávajícího HW.
2. Rozprostřít výkon mezi více počítačů.

1.4.2 Navýšení výkonu stávajícího HW

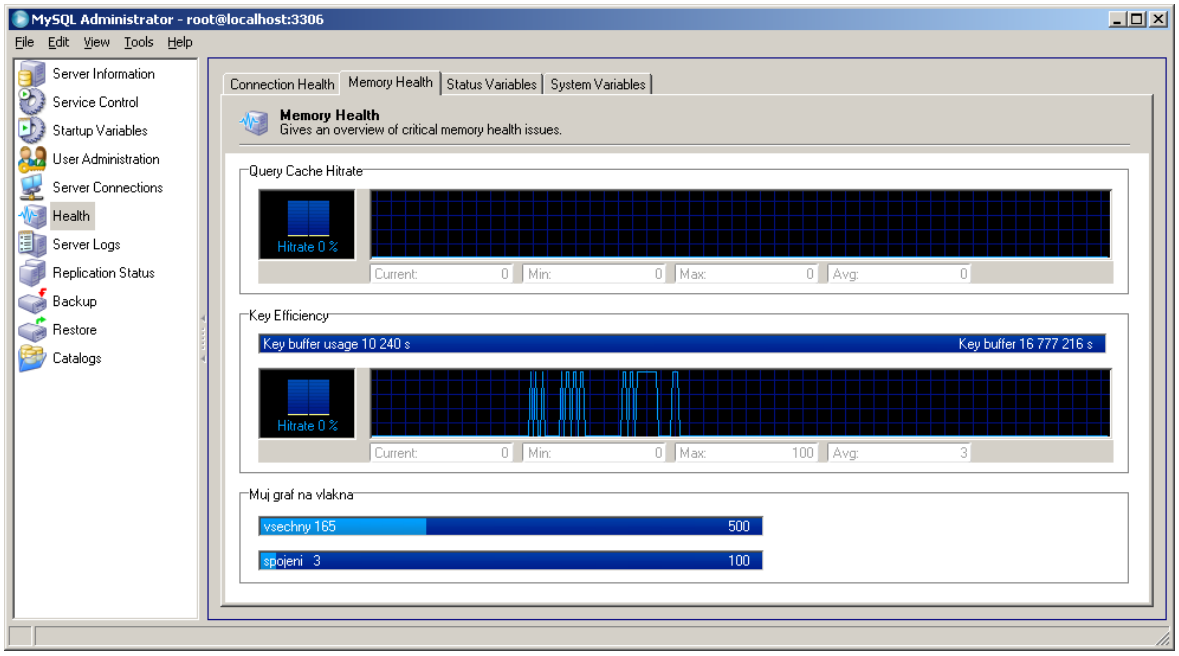
První variantu použijeme pouze v případě, že propustnost sítě je stále dostačující. Asi by nemělo smysl vyměnit HW a zjistit, že se nic nestalo, protože úzkým hrdlem byla síť. Předpokládejme tedy, že síť má k limitu ještě daleko, ale HW silně zaostává. Reakcí by bylo vzít spoustu peněz a jít koupit zcela nový HW. To je jistě ideální řešení, ale udělat to

bez rozmyslu, není dobré. Na problémy se musíme podívat blíže. Je potřebné zjistit, co a proč zaostává. Pokud koupíme nový HW a pak zjistíme, že se chová stejně, tak je něco špatně. Například má naše databáze omylem nastaven limit pro počet simultánních připojení na obrovské číslo. Cache procesů naopak velmi nízko a podobně. Vždy je žádoucí použít nějaký nástroj, který nám řekne a ukáže, co se v PC děje za provozu. Použít k tomu můžeme například některý ze správců úloh (Obr. 54).



Obr. 54. Windows XP - správce úloh.

V něm lze nastavit další sloupečky, které nám pomohou odhalit problémy s vytížením. Existují však nástroje, které se vysloveně specializují na databázový stroj MySQL. Například již zmiňovaný PhpMyAdmin nebo sám mysql, dokáže zobrazovat základní informace o hodnotách parametrů běžícího serveru. Ideálním případem jsou však nástroje, které umí i vizualizovat vytížení jednotlivých částí MySQL. Příkladem budiž MySQL Administrátor [15] (Obr. 55).



Obr. 55. MySQL Administrátor.

Tento program je v současnosti nahrazen programem MySQL Workbench [16]. Nicméně i starší verze MySQL Administrátora plně dostačují. Program umí dynamicky zobrazovat pohyb hodnot v čase. Umí také vytvářet nové grafy na existující parametry. My jsme si například vytvořili panel se dvěma horizontálními grafy. Jeden zobrazuje celkový počet vláken MySQL a druhý počet vláken, které používají přihlášení uživatelé. Na internetu je celá řada takovýchto programů. Stačí si vybrat ten, který nám bude nejlépe vyhovovat. Řekne nám, jaké je vytížení **CPU**, RAM, I/O operaci. Až na základě analýzy systému se rozhodneme a provedeme patřičný zásah. Zjistíme například, že CPU pracuje na 40%, ale disky na 100%. RAM není zaplněná ani z poloviny. To je signál, že něco není v pořádku. CPU nás nebrzdí a RAM je dostatek. Úzkým hrdlem jsou tedy I/O operace. Disky jsou plné, nebo jsou různé cache MySQL nastaveny na tak nízkou hodnotu, že se RAM prakticky nevyužívá. Stačí navýšit softwarové hodnoty. Načítat do pamětí více informací a výkon lépe rozprostřít mezi systémové zdroje. HW může zůstat původní.

Časem ovšem nastane situace, kdy je výměna komponentů nebo celého HW nezbytná. I zde je třeba jednat rozumně. Musíme znát naši databázi. Její charakter. Ideální je ušít HW přímo na naší databázi. Například volba CPU. Použít dvě či čtyři jádra? Nebo rovnou víc procesorů? Jakou taktovací frekvenci, množství vyrovnávací paměti? Vše závisí na databázi. Pokud naše databáze používá server jako jediná (dedikovaný server), je rozumné

použít CPU s co nejvyšším taktem. Počet jader je irelevantní. Samozřejmě za předpokladu, že databáze není naprogramovaná jako paralelní aplikace, ale pracuje pseudoparalelně. Naopak, když na serveru pojede víc databází, nebo víc služeb, je výhodné použít více jader i s nižším taktem. Využijeme paralelního provozu jednotlivých aplikací. Paměť RAM je také důležitá. Té není nikdy dost. Nicméně opět nemá smysl kupovat 8, 16 nebo 32 GB RAM, když OS pracuje na 32 bitech. Nebo máme zastaralý typ databáze a ten nepodporuje víc než 4 GB RAM.

Asi největším problémem jsou standardně disky. Těm je potřeba věnovat velkou pozornost. Například použít disková pole RAID. Typu nejlépe 10. Dnes také můžeme, za zlomek ceny, dosáhnout i lepších výsledků s použitím **SSD** disků. Vždy záleží na velikosti úložného prostoru. SSD jsou zatím výhodné pro malé kapacity. Pro profesionální použití je ještě možné použít RAM disky. Zde je pak přenosová rychlost až 1 GB/s. Nicméně cena je značná. Právě cena je limitujícím faktorem, který nás nutí přemýšlet nad tím, jak optimalizovat HW.

1.4.3 Rozprostření výkonu na více počítačů

Jinou možností, jak zvýšit výkon naší databáze, je rozložit ji na segmenty. Například tak, že vlastní démon mysqld poběží na jednom PC a data se budou ukládat na separátní server či diskové pole. MySQL umí alokovat přímo diskový oddíl a de facto obejít režie operačního systému. Platí to pro tabulky InnoDB. Takový oddíl pak řídí sám ovladač databáze, a proto je přesně optimalizován. OS tento diskový oddíl vidí jako nealokovaný.

Při vrstvení výkonu opět záleží na charakteru databáze. Pokud převládají výběrové dotazy, je možné podstatné navýšení výkonu pomocí replikace dat. Replikace je technika, kdy MySQL zkopíruje databázová data na pobočné počítače. Hlavní řídící počítač pak vytváří synchronizační soubory. Tyto soubory jsou pak čteny replikanty, kteří se podle nich synchronizují s originální databází. K databázi pak klient přistupuje podle typu požadavku. Některý klient nemá v úmyslu do databáze zapisovat. Nebo ani nemá oprávnění tak činit. Pak ho klient směřuje na jeden z pobočných systémů. Nebo na ten, který je nejméně vytížen. Nevýhodou replikace je fakt, že v případě požadavku na zápis do databáze musí být provoz vždy přesměrován na hlavní stroj. Replikace je pouze jednosměrná. Replikované stroje pak mohou být umístěny kdekoliv. Klidně v jiném státě. Výhodou je podstatné navýšení výkonu v provádění výběrových příkazů SQL. Samozřejmě také

rozprostření síťového provozu. Replikaci lze také chápat jako jisté zálohování dat. Tedy navýšení bezpečnosti.

Replikace byla poslední kapitolou teoretické části. Nyní nám nic nebrání, abychom se vrhli na praktickou část a implementovali do našeho klienta z bakalářské práce některé technologie uvedené v teoretické části.

II. PRAKTICKÁ ČÁST

2 VÝBĚR POTENCIALNÍCH ÚPRAV

V teoretické části jsme si řekli o víceuživatelském přístupu do databáze. S tím souvisí problematika autentizace a autorizace. Tedy definice přístupových práv na databázi a data v ní uložená. Dále jsme se celkem podrobně zabývali transakcemi a ochranou dat. Nakonec jsme si řekli něco málo o navyšování výkonu ze strany HW. Než začneme našeho klienta vylepšovat, tak si musíme položit otázku. Zapracujeme do klienta všechny výše uvedené technologie? Záleží na charakteru databáze, kterou klient obhospodařuje.

1. Potřebuje náš klient realizovat víceuživatelský přístup? Na to je snadná odpověď. Jednoznačně ano. Chceme přece nabízet naše data širokému okruhu uživatelů.
2. Potřebujeme transakce? V případě naší databáze nejsou transakce vůbec nutné. Data nejsou nijak kritická. Naše databáze nemusí řešit křížení záznamu přes tabulky, neboť používá na data pouze jedinou tabulku. Referenční integrita nás tedy nebude trápit. Rovněž nebudeme potřebovat žádné atomické operace. Proto nebudeme transakce do klienta integrovat. Jedná se ovšem o tak zajímavé téma, že si v praktické části ukážeme postup, jak upravit klienta, aby transakce podporoval.
3. Zabezpečení dat a datového přenosu. Bude nám vadit, když nám někdo data přepíše nebo smaže? Mně ne. Vám ano. Jsou to jenom obaly na CD. Nic víc. Žádná soukromá nebo citlivá data. Navíc jsou na internetu redundantní. A mnohokrát. Rovněž zabezpečení přenosového kanálu je nepotřebné. Databáze je veřejná. Šifrování by bylo kontraproduktivní. Náš klient je samozřejmě touto technologií vybaven. Ukážeme si, že stačí modifikovat 3 řádky formuláře a náš klient začne navazovat SSL spojení jako divý. Na to, abychom ho přeprogramovali na SSL, nám stačí 10 vteřin. Takže odpověď na tuto otázku je také negativní.
4. Navyšování výkonu klienta je zajímavá otázka. Bohužel výkon nelze dost dobře testovat. Naše databáze je cvičná a i kdyby měla 1000 záznamů, tak bychom při dnešní výpočetní síle nic optimalizovat nemuseli. Jakékoliv optimalizační techniky, které aplikujeme, nedokážeme relevantně posoudit. Jeden aspekt k velmi razantnímu nárůstu výkonu náš klient přece jenom obsahuje. Dokonce tak zásadní, že by mohl případně ovlivnit naše rozhodnutí ohledně transakcí. Je jím sama struktura hlavní tabulky ***av*** databáze ***audio_video***. Pro jednoduchost je navržena

jako celek. Pomineme-li neoptimální strukturu tabulky, tak jedna položka je v této tabulce opravdu velice nešťastná. Jsou v ní dvě pole typu BLOB. Do nich ukládáme obrázky v tisknutelné kvalitě. To není vůbec dobré. Ne tak ani z hlediska tabulky, jako z hlediska rychlosti prohledávaní této tabulky. Ideální by bylo oddělit obrázky a ukládat je do separátní tabulky nebo do externích souborů. Ještě lepší by bylo, mít dva typy obrázků. Jeden pro náhled a jeden pro tisk. Obrázek s vysokou kapacitou by byl načítán až při potřebě tisku. Tím by se velice odlehčilo síti. De facto se jedná pouze o zvýšení rychlosti klienta při načítání dat. Databáze však byla špatně navržena úmyslně. V bakalářské práci [8] je na to poukázáno. Bylo tak učiněno čistě pro jednoduchost. V diplomové, ani v bakalářské práci, není řešena struktura databáze. Řešíme zde vztah klient-databáze a programování klienta. Návrh velmi jednoduché databázové struktury byl proto namístě. Navíc rychlost sítí je tak vysoká, že náš klient tím příliš netrpí. Jedinou slabostí takto navržené tabulky je její procházení ve stavu, kdy by byla naplněna velkým počtem záznamů. Rozhodnutí o modifikaci klienta je poměrně těžké. Nicméně primárně jsme neřešili návrh databázové struktury. Nebylo to cílem ani bakalářské, ani diplomové práce. Proto nebudeme databázovou strukturu modifikovat. Tedy ani klienta. Zaměříme se pouze na jemné vyladění víceuživatelského provozu.

Položené otázky nám pomohly specifikovat rozsah potřebných úprav, které integrujeme do stávajícího klientského programu.

2.1 Víceuživatelský přístup do databáze

Jediným zásahem, i když podstatným, bude naprogramovat klienta tak, aby dokázal identifikovat uživatele a jejich oprávnění. Dále se musíme postarat o souběžné vykonávání dotazů více nezávislými klienty. Naučíme program registrovat uživatele. Na základě registrace bude uživateli povoleno vkládat, modifikovat i mazat záznamy. To vše ve spolupráci s konfiguracemi. Navíc nesmíme porušit strukturu již napsaného programu. Databáze MySQL má velice propracovaná přístupová práva. K řešení našeho problému existují dvě cesty.

1. Použijeme k autentizaci server MySQL a tabulku ***user***.
2. Použijeme k autentizaci vlastní tabulku ***cfg*** a vlastní systém autentizace.

Nyní se můžeme zamyslet nad otázkou, proč používat vlastní systém pro ověřování uživatelů, když MySQL má tak výborný autentizační protokol. Podívejme se na to z pohledu uživatele. Pokud se přihlásíme k nějakému serveru, který není zcela otevřen, tak se musíme zaregistrovat. Například proto, abychom si rozšířili funkcionalitu. Třeba nám bude povolen vstup do zákaznické sekce, do které se jinak nedostaneme. Ať tak či onak, bude registrace provedena systémem automaticky nebo na základě naší žádosti ručně.

Ruční registrace znamená, že nám na naši žádost bude přiděleno přihlašovací jméno a heslo nějakým oprávněným uživatelem. Což může nějakou dobu trvat. Automatická registrace proběhne ihned. Co je pro databázi lepší? Samozřejmě ruční inicializace nového zákazníka. Proč? To je otázka týkající se bezpečnosti databáze. Při ručním vložení nového zákazníka se o tuto akci postará oprávněná osoba. Tedy osoba s privilegiemi GRANT. Zpravidla správce systému. To je čisté řešení, při kterém nikdy není k dispozici žádný volný přístup k databázi. Kdo však vloží uživatele do databáze při automatické registraci? V databázi musí být nějaký univerzální systémový uživatel s právem vytváření nových uživatelů. A to je bezpečnostní riziko. Ruční vkládání nových uživatelů pro nás nepřipadá v úvahu. Chceme rychlou reakci, abychom si neodradili zákazníka. Ručně to lze aplikovat na podnikových nebo třeba bankovních aplikacích. Tedy tam, kde se nepočítá s příliš vysokým registračním potenciálem. Nebo je na registraci dostatek času.

Nezbývá nám tedy, než upravit klienta tak, aby umožnil nového zákazníka zaregistrovat automaticky. Přeprogramujeme našeho klienta z bakalářské práce. Po úpravě se bude chovat jako webová aplikace.

2.2 Autentizace klienta na úrovni databázového stroje MySQL

Pokusme se naučit databázi MySQL registraci. Prostudujeme-li si uživatelský manuál [14] tak zjistíme, že MySQL umí registraci. Existuje možnost vytvoření speciálního uživatele. Tomuto uživateli nadefinujeme na úrovni tabulky ***user*** jediné právo. Právo vytvářet uživatelské účty CREATE_USER_PRIV (Obr. 56).

☐	root	127.0.0.1	Ne	ALL PRIVILEGES	Ano	
☐	root	localhost	Ne	ALL PRIVILEGES	Ano	
☐	test	localhost	Ne	CREATE USER	Ne	

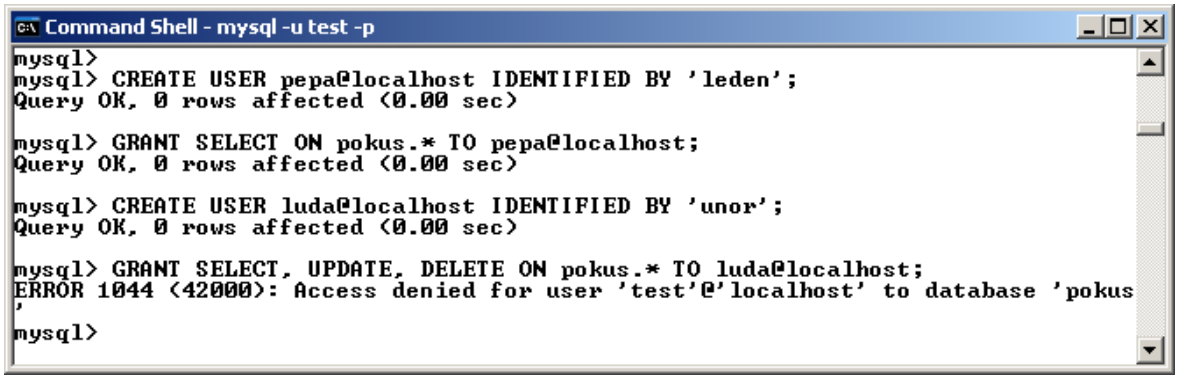
Obr. 56. PhpMyAdmin - výpis tabulky user.

Založili jsme uživatele *test*, který se může přihlásit z počítače „localhost“. Tento uživatel má jediné právo. Zakládat nové uživatele. To však nestačí. Noví uživatelé se musí dokázat nejenom připojit k MySQL, ale také do nějaké databáze. Musíme proto ještě do tabulky ***db***. Tam povolíme uživateli *test* manipulace s databází ***pokus***. Nyní ale nastává chvíle, kdy si dobře rozmyslíme, co všechno budou moci budoucí uživatelé s databází provádět. Tedy uživatelé, které vytvoříme prostřednictvím účtu *test*. Noví uživatelé mohou získat pouze taková oprávnění, jako má *test*. Nebo pouze některá z nich. Avšak žádná navíc. Povolíme tedy uživateli *test* privilegia na databázi ***pokus*** (Obr. 57).

←T→	Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv
☐ ✎ ✕	localhost	phpmyadmin	pma	Y	Y	Y	Y	N	N	N
☐ ✎ ✕	%	audio_video	pripojovac	Y	Y	Y	Y	N	N	N
☐ ✎ ✕	localhost	pokus	Dagmar	Y	Y	Y	Y	N	N	N
☐ ✎ ✕	localhost	pokus	Eda	Y	Y	Y	Y	N	N	N
☐ ✎ ✕	localhost	pokus	test	Y	Y	N	N	N	N	Y

Obr. 57. PhpMyAdmin - výpis tabulky db.

Zde vidíme, že *test* má povolen SELECT a INSERT. Navíc ještě GRANT_PRIV. Nemusíme se privilegia GRANT bát. To se novému uživateli již nekopíruje. Nyní není problém, prostřednictvím účtu *test*, zaregistrovat nového uživatele. Můžeme mu přidělit práva na databázi ***pokus*** (Obr. 58). Účet *test* je tedy univerzální uživatel, přes kterého dokáže klient registrovat nové uživatele. Jeho jméno a heslo by měl znát právě pouze klient!



Obr. 58. Mysql – registrace pomocí MySQL.

Uživatel *test* založil dva nové účty. Účet *pepa* a *luda*. Oba bez diakritiky. Pepovi povolil pouze SELECT na všechny tabulky v databázi ***pokus***. Luďovi se však pokusil přidělit víc privilegii, než má sám. MySQL mu to nepovolila. Zde je výpis z tabulky ***db*** (Obr. 59).

	Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv
	localhost	phpmyadmin	pma	Y	Y	Y	Y	N	N	N
	%	audio_video	pripojovac	Y	Y	Y	Y	N	N	N
	localhost	pokus	Dagmar	Y	Y	Y	Y	N	N	N
	localhost	pokus	Eda	Y	Y	Y	Y	N	N	N
	localhost	pokus	pepa	Y	N	N	N	N	N	N
	localhost	pokus	test	Y	Y	N	N	N	N	Y

Obr. 59. PhpMyAdmin - aktualizovaný výpis tabulky db.

Všimněme si, že *pepa* již nemá privilegium GRANT_PRIV. Uživatel *test* může nejenom zakládat účty, ale také je likvidovat příkazem DROP USER. Odstraní se automaticky i záznam z tabulky ***db***.

2.3 Autentizace klienta na úrovni databázové tabulky audio_video

Druhou možností je ponechat tabulku ***user*** a ***db*** tak jak je. Vytvořit si vlastní tabulku v databázi ***audio_video*** a v ní uchovávat přihlašovací údaje zákazníků. Modifikace databáze ***user*** a ***db*** proběhne pouze jednou. I zde musíme vytvořit univerzálního uživatele. Pomocí tohoto uživatele se budeme připojovat na databázi a obhospodařovat interní uživatelské účty. Autentizaci uživatelů budeme řídit přes tabulku ***cfg*** pomocí klienta. Před tím než začneme, provedeme import databáze ***audio_video***. Ta obsahuje dvě tabulky. Tabulku ***av*** a ***cfg***. Importování je popsáno v bakalářské práci [8]. Import je nejlépe provést

pomocí programu PhpMyAdmin. Tabulka ***av*** obsahuje vlastní data pro obaly audio nosičů. Tabulka ***cfg*** pak konfigurace k jednotlivým účtům. Celá databáze ***audio_video*** je na přiloženém CD v adresáři MYSQL. Soubor se jmenuje ***audio_video.sql***.

3 MODIFIKACE KLIENTA

Rozhodli jsme se pro variantu, kdy se klient bude chovat jako webová aplikace. Měli bychom upravit MySQL a klienta takto:

1. Vložení univerzálního uživatele do databáze MySQL.
2. Ihned po spuštění klienta budou k dispozici data, ale pouze v režimu SELECT.
3. Klient bude nabízet přihlášení i registraci uživatelů.
4. Přihlášení uživatelé budou mít rozšířená oprávnění (UPDATE, INSERT, DELETE).
5. Uživatele budou konfigurovatelní.
6. Maximální stáří neaktivního účtu bude 1 měsíc. Poté bude automaticky odstraněn.
7. Klient se po určité době nečinnosti sám odpojí od databáze.
8. Uživatelské přístupy bude řídit tabulka ***cfg*** z databáze ***audio_video***.
9. Klient bude přizpůsoben víceuživatelskému provozu.

3.1 Modifikace přístupových práv MySQL

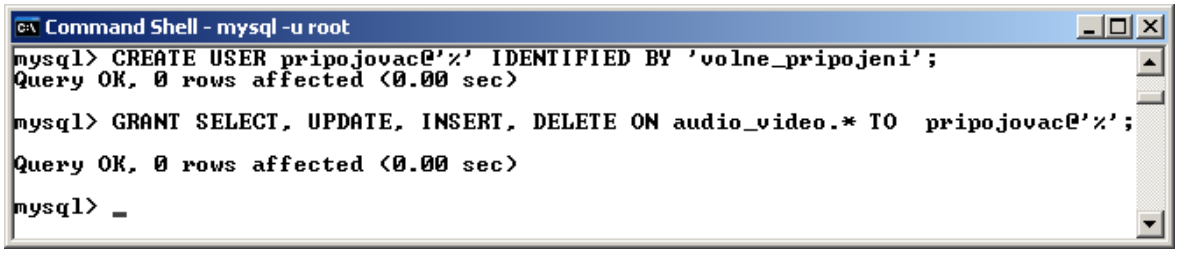
V databázi MySQL si vytvoříme univerzálního uživatele. Nazveme ho například *pripojovac*. Jako heslo použijeme „*volne_pripojeni“*. Nebudeme používat diakritiku.

```
CREATE USER pripojovac@'%' IDENTIFIED BY 'volne_pripojeni';
```

Nezapomeneme nastavit přístup odkudkoliv (%). Dále nastavíme tomuto uživateli privilegia SELECT, INSERT, UPDATE a DELETE v tabulce ***db*** na databázi ***audio_video*.** Pro všechny tabulky.

```
GRANT SELECT, UPDATE, INSERT, DELETE ON audio_video.* TO pripojovac@'%';
```

Přes tohoto uživatele se budeme připojovat do databáze (Obr. 60).

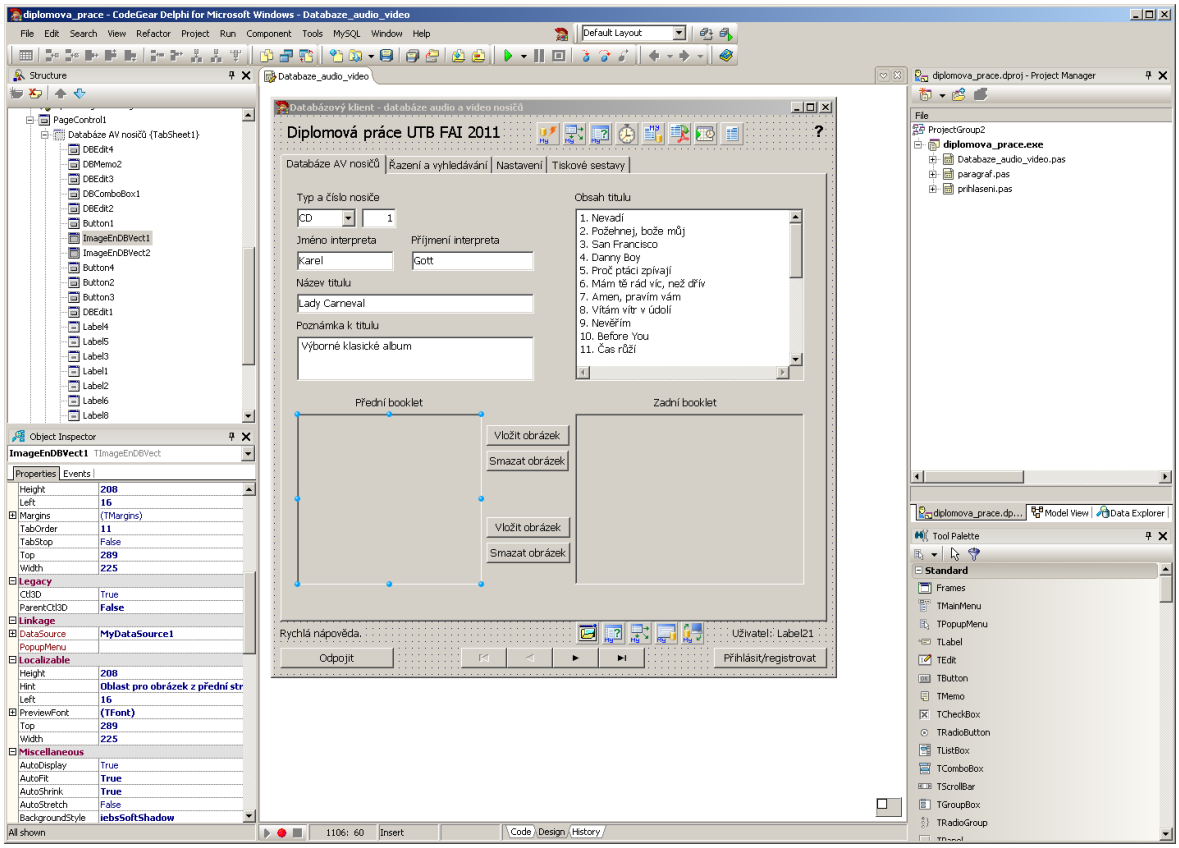


Obr. 60. Mysql - nastavení univerzálního uživatele.

Z hlediska databáze MySQL nemá *pripojovac* nijak zvláštní privilegia. Úpravy může dělat pouze ve své databázi ***audio_video***. Pomocí klienta si budeme řídit jeho chování sami. Samozřejmě, že kdokoliv se znalostí univerzálního hesla může použít jiného klienta a udělat nám v datech paseku. Univerzální uživatel je tedy jisté bezpečnostní riziko. I když má minimální privilegia.

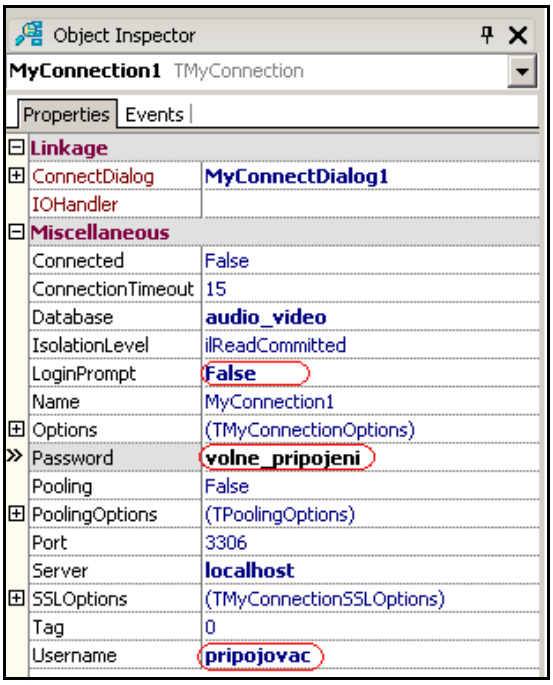
3.2 Úprava klienta – automatické přihlášení do režimu SELECT

Připomeňme, že program klienta je napsán v Delphi. Konkrétně v CodeGear RAD studiu 2007 (Obr. 61). Tento program je nyní nahrazen novou verzí, dostupnou na stránkách Embarcareto [17].



Obr. 61. Delphi 2007 – ukázka pracovní plochy programu.

Nejprve upravíme přihlašovací strategii klienta. Nebudeme zobrazovat přihlašovací formulář. Všechny potřebné přihlašovací údaje naprogramujeme napevno. Potřebujeme specifikovat přihlašovací jméno, heslo, komunikační port a server, na který se chceme napojit. To vše změníme najednou v jediné komponentě ***MyConnection1*** (Obr. 62).



Obr. 62. Delphi - modifikace připojení

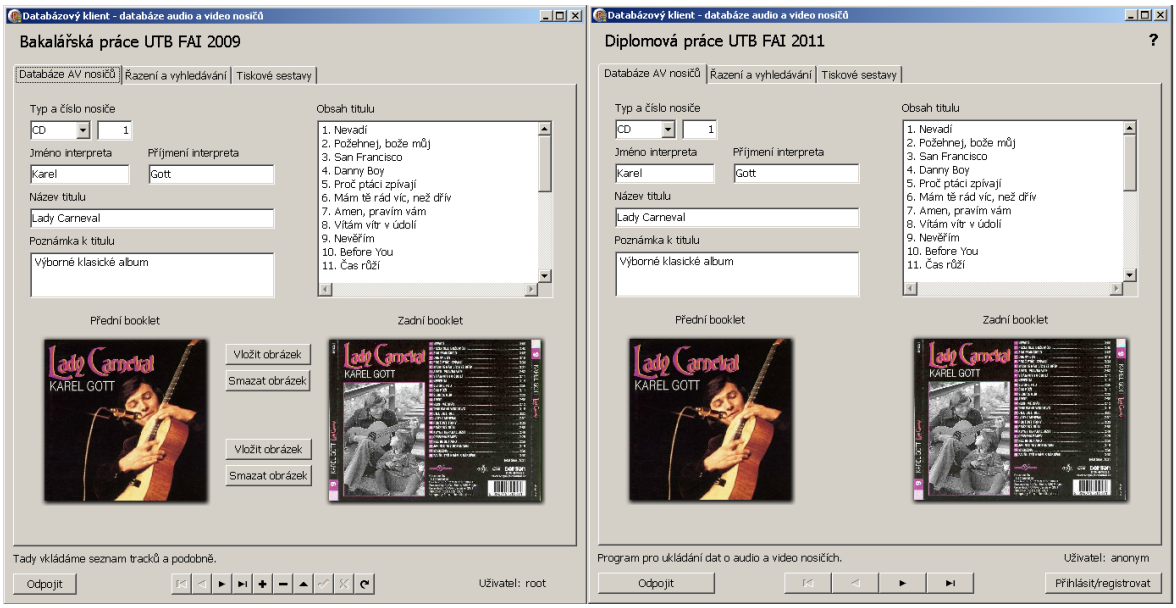
Položka ***LoginPrompt*** zamezí volání přihlašovacího formuláře. Ten již není potřeba. Do parametru *Password* a *Username* nastavíme údaje dle obrázku. Je to náš univerzální uživatel, přes kterého budeme s databází komunikovat. Znamená to tedy, že v tomto stavu jsou všechny přihlašovací informace zadány napevno. Jak IP adresa, tak port a název databáze. Pokud by byla databáze replikovaná, bylo by možné provést náhodný výběr jednoho z dostupných serverů. Stačilo by vložit obslužný kód výběru do metody ***BeforeConnect*** komponenty ***MyConnection1***.

Dále je potřeba upravit zdrojový kód tak, aby klient reagoval na změny, které vyvolá automatické odpojení od databáze při dané době nečinnosti. Kompletní zdrojový kód je samozřejmě na přiloženém CD. Soubor se jmenuje ***databáze_audio_video.pas***. Především musíme skrýt některé ovládací prvky formuláře, které nemají vazbu na komponentu ***MyConnection1***. Jedná se o 4 tlačítka pro vkládání a mazání obrázků. Upravíme dvě metody.

```
Tform4.nacteni_cfg;

TForm4.FormCreate(Sender: TObject);
```

Potřebujeme-li skrýt ovládací prvek, stačí nastavit parametr ***Visible*** konkrétního prvku na ***false***. Pro zobrazení pak na ***true***. Naše čtyři tlačítka mají název ***Button1-4.***

Na obrázku (obr. 63) je podoba přihlášeného klienta před a po změně.



Obr. 63. Klient – úprava. Vlevo před změnou, vpravo po změně.

3.3 Úprava klienta – přihlášení a registrace

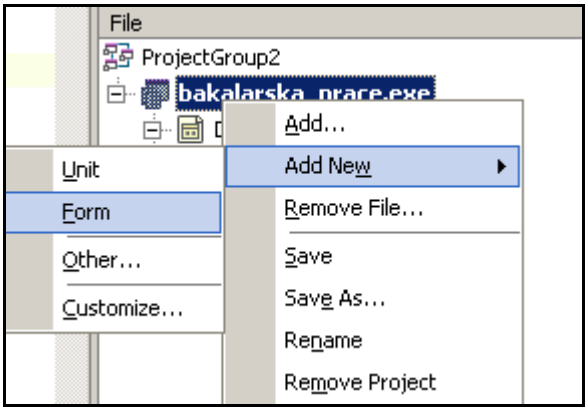
Zde již budeme muset zasáhnout do zdrojového kódu citelně [6], [7]. Jednak vytvoříme přihlašovací a registrační formulář a také bude nutné upravit tabulku ***cfg*** v databázi ***audio_video***. Přidáme do ní některé další sloupečky (Obr. 64).

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	cfg_id	int(10)		UNSIGNED	Ne		auto_increment	
☐	id_user	varchar(100)	utf8_unicode_ci		Ne			
☐	cfg_pas	char(41)	utf8_unicode_ci		Ano	*NULL*		
☐	cfg_id1	int(1)		UNSIGNED	Ne			
☐	cfg_id2	tinyint(1)			Ne			
☐	cfg_id3	tinyint(1)			Ne			
☐	cfg_id4	smallint(5)		UNSIGNED	Ne			
☐	cfg_id5	tinyint(1)			Ne			
☐	cfg_id6	tinyint(1)			Ne			
☐	cfg_id7	date			Ne			

Obr. 64. PhpMyAdmin - modifikovaná tabulka cfg.

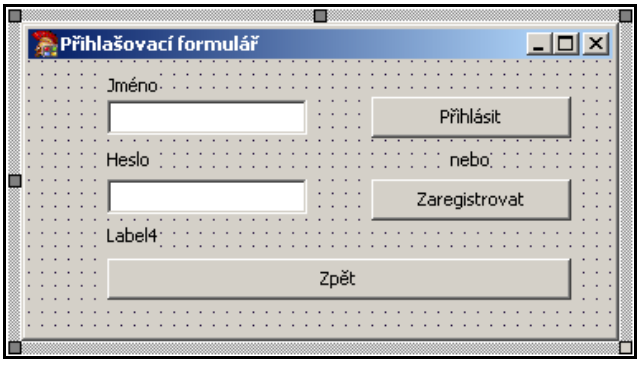
Konkrétně jsme přidali položky `cfg_pas` pro vložení hesla a `cfg_id7` pro vložení data změny. Datum se bude modifikovat vždy, když se uživatel přihlásí. Klient sám pak bude kontrolovat překročení stáří uživatele. Pokud se uživatel nepřihlásí do jednoho měsíce, bude automaticky odstraněn. U položky `cfg_pas` nezapomeneme na volání funkce PASSWORD. Chceme, aby bylo heslo jednosměrně šifrované a nečitelné.

Nyní přistupme k naprogramování přihlašovacího formuláře. Přiřadíme si k existujícímu projektu ***Bakalářská_práce.exe*** další formulář (Obr. 65). Nazveme si ho *prihlaseni* (bez diakritiky).



Obr. 65. Delphi - vložení nové jednotky.

Získáme formulář, na který si vložíme potřebné komponenty pro přihlášení nebo registraci. Může vypadat třeba takto (Obr. 66).



Obr. 66. Delphi - přihlašovací formulář.

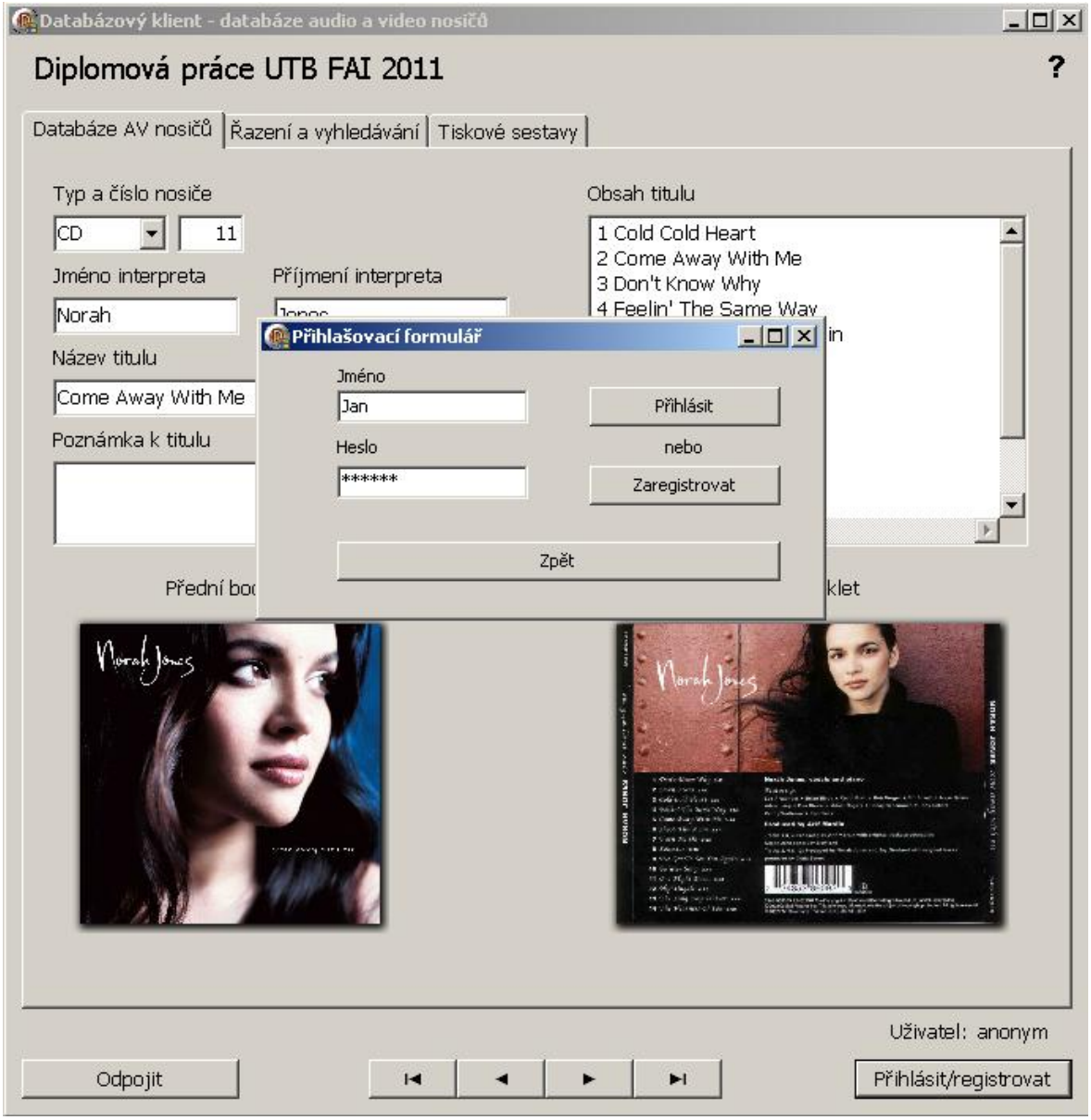
Tento formulář má svůj vlastní kód. V něm naprogramujeme metodu ***FormActivate***. Zabezpečíme tím, že si formulář nebude pamatovat jednou zadané hodnoty (Obr. 67).

Dále si v komponentě, ve které zadáváme heslo, nastavíme masku na hvězdičku. Parametr se jmenuje ***PasswordChar***. Další jednoduchý kód pro ovládání tlačítek si zde již vypisovat nebudeme. Nalezneme ho na přiloženém CD v souboru ***prihlaseni.pas***.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
Label204.Visible := false;
Label204.Caption := '';
Edit1.Text:='';
Edit2.Text:='';
Edit1.SetFocus;
end;

function Tform1.kontrola;
begin
Result := 0;
if form1.Edit1.Text = '' then
      Begin
      Showmessage('Zadej prosím přihlašovací nebo registrační
      form1.Edit1.SetFocus;
      Result := -1;
      end
```

Obr. 67. Delphi - ukázka kódu přihlašovacího formuláře.

Je možné se s formulářem hodně vyhrát. Přidáme kontrolu zadání jména a hesla. Jméno a heslo budeme vyžadovat vždy. Stejně tak jejich minimální délku 3 a 6 znaků. Nikdy nepovolíme založit uživatele bez hesla. Kdo má zájem, může ještě při registraci kontrolovat minimální složitost hesla a další jemné detaily. Po vyplnění formuláře se pošlou hlavní aplikaci informace o akci, kterou má provést. Tedy nic, přihlášení nebo registraci uživatele. Naprogramujeme obslužnou rutinu tlačítka pro přihlášení, registraci i následné odhlášení uživatele. Vlastní formulář pak voláme z hlavní aplikace metodou ***FormX.ShowModal***. Ta zajistí, že po dobu práce s přihlašovacím formulářem nebude hlavní aplikace dostupná. Na obrázku je běžící aplikace s aktivním přihlašovacím formulářem (Obr. 68).



Obr. 68. Klient – přihlášení.

3.4 Úprava klienta – kód registrace

Náš klient musí nějak komunikovat s tabulkou ***cfg*** v databázi ***audio_video***. V případě registrace potřebujeme odeslat do této tabulky jméno a heslo nového uživatele. Je třeba také zkontrolovat, jestli již podobné jméno neexistuje. Ukážeme si jednu možnost, jak komunikovat jednorázově se serverem MySQL. MyDAC k tomuto účelu vytvořil komponentu ***TMyCommand***. Vložíme ji někde na formulář hlavní aplikace. Poté ji stačí napojit na ***MyConnection1*** a zavolat ji z hlavního formuláře. SQL kód pak vložíme takto:

```
1 MyCommand1.SQL.Clear;
2 MyCommand1.SQL.Text := 'INSERT INTO audio_video.cfg (cfg_id, id_user,
  cfg_pas, cfg_id1, cfg_id2, cfg_id3, cfg_id4, cfg_id5, cfg_id6, cfg_id7)
  VALUES (NULL, "' + form1.Edit1.Text + '", PASSWORD("' +
  form1.Edit2.Text + '"), "1", "0", "1", "10", "0", "1", NOW());';
3 try
4     MyCommand1.Execute;
5 except
6     ShowMessage('Uživatel ' + form1.Edit1.Text + ' již existuje.');
7 end;
```

Jedná se o jednoduchý kód, který nejprve vymaže obsah kontejneru ***MyCommand1.SQL***. Poté do parametru *text* vložíme potřebný příkaz SQL. Nakonec ho vykonáme. ***MyCommand1.execute*** je proveden v chráněném módu. Kdyby se stala nějaká chyba, tak aplikace nespadne. Toto je pouze fragment obslužného kódu registrace. Celý kód je na přiloženém CD.

3.5 Úprava klienta – kód přihlášení

Následující ukázka kódu řeší přihlášení uživatele. Přístup je trošku jiný. Použijeme komponentu ***MyQuery2***. Právě přes tuto komponentu můžeme trvale manipulovat s daty tabulky ***cfg***. Výsledek dotazu je uložen do komponenty ***Myquery2***. Dá se pak hezky zpracovávat.

```
1 Myquery2.Close;
2 Myquery2.SQL.Clear;
3 Myquery2.SQL.Add('SELECT * FROM cfg WHERE id_user = "' +
  form1.Edit1.Text + '" AND cfg_pas = PASSWORD("'+ form1.Edit2.Text
  +'");');
4 Myquery2.Execute;
```

Nejprve uzavřeme ***Myquery2***. Pak smažeme případný existující dotaz. Poté vložíme dotaz nový a vykonáme ho. Je nám vrácen výsledek. Na jeho základě pak zjišťujeme, zda přihlašovaný uživatel existuje nebo ne. Pokud není v kontejneru MyQuery2 žádný záznam tak uživatel neexistuje. Pokud je vrácen právě jeden záznam, je uživatel nalezen a přihlášen. Počet aktuálních záznamu v kontejneru MyQuery2 zjistíme takto:

```
1 if myquery2.RecordCount = 1 then
2 Begin
3     // načte konfigurace přihlášeného uživatele
4     nacteni_cfg;
5     // aktualizuje uživateli nové datum přístupu
6     MyCommand1.SQL.Clear;
7     MyCommand1.SQL.Text := 'UPDATE audio_video.cfg \
      SET cfg_id7 = NOW() WHERE id_user = "' + form1.Edit1.Text + '"';
8     try
9         MyCommand1.Execute;
10    except
11        showmessage('Nepodařilo se aktualizovat datum přihlášení.');
12    end;
13 End
14 else
15 Begin
16    showmessage('Uživatel nebyl nalezen.')
17 End;
```

Ve výpise programu je také aktualizace data přihlášení. ***MyCommand1.Execute*** je proveden v chráněném módu. Pokud nebude příkaz vykonán, bude zobrazena zpráva v těle události except.

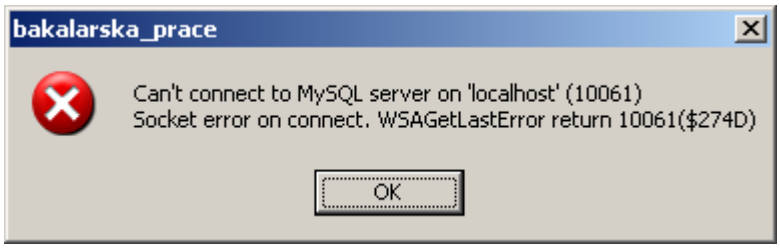
3.6 Úprava klienta – kontrola expirace účtu

Následující kód se postará o odstranění uživatelů, kteří se nepřihlásili déle jak 31 dnů. Opět použijeme komponentu ***MyCommand1***. Tato procedura je celkem užitečná. Spousta uživatelů totiž zapomene své přihlašovací heslo. Místo toho aby ho vyhledali nebo požádali o nové heslo správce databáze, provedou novou registraci. Počet neaktivních účtů může být časem obrovský. V praxi bychom postupovali citlivěji. Účet bychom zrušili po mnohem delší době nečinnosti. Také bychom informovali uživatele o zrušení účtu prostřednictvím mailu. Možností je celá řada. Následující výpis ruší účet po 31 dnech:

```
1 MyCommand1.SQL.Clear;
2 MyCommand1.SQL.Text := 'DELETE FROM cfg \
  WHERE (DATEDIFF(NOW(),cfg_id7)) > 31;';
3 try
4     MyCommand1.Execute;
5 except
6     showmessage('Nepodařilo se odstranit staré účty');
7 end;
```

3.7 Úprava klienta – odchycení chybových hlášení

Při komunikaci klienta s databází MySQL může docházek k chybám. Nejedná se jenom o chyby na straně klienta, ale také o informace, které databáze posílá klientovi. Na straně klienta jsou pak tato hlášení zobrazována ve stejné podobě, v jaké je databáze klientovi poslala. Totéž se týká výjimek modulu MyDAC. Na obrázku (Obr. 69.) je ukázka přihlášení aplikace v okamžiku, kdy nebyla databáze v provozu.



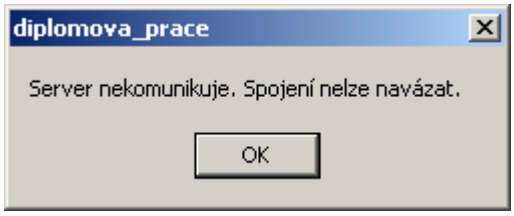
Obr. 69. Klient – chyba připojení.

Klienta lze lokalizovat do českého jazyka různými způsoby. Můžeme odchytávat zprávy a podle čísla je překládat do srozumitelné podoby (Obr. 70). Kód je zapsán do metody ***MyConnection1Error*** komponenty ***Myconncetion1***.

```
procedure TForm4.MyConnection1Error(Sender: TObject; E: EDAError;
  var Fail: Boolean);
begin
//showmessage (inttostr(e.ErrorCode));
if e.ErrorCode = 2003 then
  Begin
1110  showmessage('Server nekomunikuje. Spojení nelze navázat.');
  Fail := false;
  End;
```

Obr. 70. Klient – odchycení chyby.

Pokud ho použijeme, dostaneme při pokusu o připojení na zastavený server MySQL následující zprávu (Obr. 71). Ta se již liší od zprávy z obrázku (Obr. 69).



Obr. 71. Klient – česká lokalizace.

Druhou možností, jak lokalizovat aplikaci do českého jazyka, je přepsat existující protokol chyb. MySQL disponuje soubory, ve kterých jsou seřazena všechna hlášení, která MySQL produkuje. Stačí tento soubor vyměnit za jiný s českým překladem. V distribuci XAMPP je česká lokalizace již dodána. Najdeme ji v adresáři C:\xampp\mysql\share\. Pro jeho aktivaci stačí spustit démona mysqld takto:

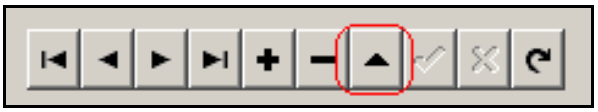
```
mysqld -L czech
```

Samozřejmě je ideální modifikovat konfigurační soubor my.ini. V něm doplnit řádek „language=czech“. Pak stačí mysqld startovat běžným způsobem. Od této chvíle budou veškerá hlášení v českém jazyce. Nesmíme ovšem zapomenout na vhodnou znakovou sadu. Jinak nebude česká diakritika zobrazována správně.

3.8 Úprava klienta – víceuživatelský provoz

V této kapitole přizpůsobíme klienta na víceuživatelský provoz. Hotový program spustíme dvakrát. Přihlásíme se do databáze a smažeme záznam v jednom klientovi a pak tentýž záznam i v druhém. Přestože nepoužíváme transakce, tak se nic moc dít nebude. Klient dostane od databáze chybovou zprávu o neexistenci záznamu. Záznam již někdo smazal těsně před námi. Stačí provést ***refresh*** a načíst platná data. Ukážeme si, jak lze v rámci MyDAC vylepšit chovaní klienta. Víc ho přizpůsobíme na víceuživatelský provoz. První věcí, kterou uděláme, bude zakázání automatické modifikace v komponentě ***MyDataSource1***. Parametr ***AutoEdit*** nastavíme na ***False***. Pokud nyní budeme chtít

modifikovat záznam, tak musíme kliknout na navigační panel a příslušné editační tlačítko. To nastaví vybraný záznam do editačního módu (Obr. 72).



Obr. 72. Klient – potvrzení modifikace.

Touto malou akcí pak můžeme přinutit konkrétní záznam vykonat příkaz ***RecordRefresh***. Tedy provést aktualizaci pouze tohoto záznamu. Uděláme to tak, že do metody ***BeforeEdit*** komponenty ***MyQuery1,*** vložíme kód ***Myquery1.RefreshRecord***. To je celé. Tento postup je výhodný jak pro databázi, tak pro přenosový kanál. Není nutné posílat rozsáhlé aktualizované sady dat. Stačí pouze záznam, se kterým máme v úmyslu pracovat. Další možnou úpravou je provádění automatické aktualizace záznamů pomocí metody ***AutoRefresh.*** Interval řídí parametr ***AutoRefreshInterval***. Opět se jedná o komponentu ***MyQuery1***. Možností je celá řada. Stačí nastudovat a vyzkoušet. Na závěr se ještě zmíníme o jednom parametru. Konkrétně o parametru ***StrictUpdate***. Pokud tento parametr vypneme, tak se při pokusu o smazání neexistujícího záznamu nic nestane. Tento záznam se smaže z vyrovnávací paměti bez toho, aby se klient pohoršoval nad tím, že již neexistuje. Je to tedy podobné chování jako u transakce.

Tímto posledním krokem bychom s úpravami klienta mohli skončit. Můžeme ještě celou aplikaci přejmenovat z bakalářské na diplomovou práci. Celý klient byl naprogramován jako výuková aplikace. Není zcela optimální. Je na něm ještě mnoho práce. Ta však již leží plně v moci čtenářů. Stačí podrobit klienta testovacímu provozu a hledat chyby v komunikaci. Prostě s ním začít pracovat, vylepšovat ho a odstranit nalezené chyby.

3.9 Transakce

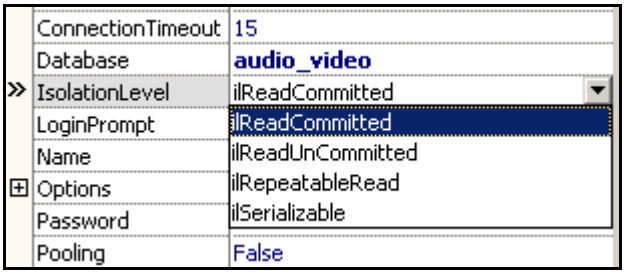
V této kapitole již nebudeme klienta měnit. Pouze předvedeme, jak by se dal upravit, aby podporoval transakce. Můžeme použít několik způsobů. Záleží to na povaze klienta. V našem případě by byla úprava snadná. Vzhledem k tomu, že pro jednoduchost byla zvolena jediná tabulka, tak transakce moc práce mít nebudou. Je to proto, že při

modifikaci jednoho záznamu není nutné implicitně volat transakci. V tabulce InnoDB je každý příkaz vykonán jako transakce. Samozřejmě pokud je přepínač AUTOCOMMIT nastaven na 1. Náš klient nemá snahu měnit záznamy hromadně. Proč taky. Jakákoliv modifikace, odstranění, či přidání záznamu proběhne jako jedna akce. Není tedy nutné jakkoliv informovat databázi, že se něco bude dít. Pokud bychom tedy opravdu chtěli provozovat transakce, tak musíme provést několik úprav jak na databázi, tak na klientovi.

V rámci databáze je nutné převést tabulky **av** a ***cfg*** na typ InnoDB. Nyní jsou typu MyISAM. Tabulky MyISAM nepodporují transakce. To je ze strany databáze vše. Příkaz pro převod tabulky je následující:

```
ALTER TABLE audio_video.av ENGINE InnoDB;
ALTER TABLE audio_video.cfg ENGINE InnoDB;
```

Na straně klienta stačí nastavit příslušnou izolaci transakcí. Jedná se o parametr ***IsolationLevel*** v komponentě ***MyConnection1*** (Obr. 73).



Obr. 73. Delphi - nastavení izolace transakce.

IsolationLevel je v modulu MyDAC defaultně nastavena na ***ReadCommited***. Víme, že tabulky InnoDB jsou optimalizovány na izolace ***RepeatableRead.*** Stupeň izolace je tedy na výběru uživatele. Můžeme nastavení ponechat, jak je. Záleží na povaze dat. Náš klient si vystačí i s nízkou izolací.

Na úrovni našeho klienta je tato úprava dostatečná. Záznamy v databázi jsou kompaktní a jejich modifikace není hromadná. Stačí tedy podpora mikrotransakcí. Co operace, to transakce. Pokud bychom potřebovali transakce sami řídit, je to také velice jednoduché. Až dosud jsme postupovali tak, že jsme se s databází spojili pomocí komponenty ***MyConnection1***. Záznamy byly načítány do komponenty ***MyQuery1.*** Spojení bylo tedy

permanentní. V případě ručního řízení transakcí je tato metoda po úpravě také možná. Chceme-li tedy ovládat transakce, tak musíme volat některé metody komponenty ***MyConnection1***. Konkrétně:

```
MyConnection1.StartTransaction;
MyConnection1.Commit;
MyConnection1.Rollback;
```

Do formuláře bychom mohli přidat tři tlačítka. Jedno pro start transakce a další dvě pak pro potvrzení či odvolání transakce. Všechna tlačítka by použila metodu ***OnClick***. Pokud bychom potřebovali vykonat několik úprav v rámci jedné transakce, stačilo by nejprve stisknout tlačítko start s metodou ***MyConnection1.StartTransaction***. Poté bychom prováděli potřebné modifikace a na závěr práce pak transakci potvrdili tlačítkem commit. V jeho metodě ***OnClick*** by byl příkaz ***MyConnection1.Commit***. Stačí tedy ručně volat tyto metody a navazující komponenty modulu MyDAC se postarají o zbytek. Platí to i pro komponenty ***MyCommand.*** Cokoliv mezi voláním metody ***MyConnection1.StartTransaction*** a ***MyConnection1.Commit (Rollback)*** je vykonáno jako transakce. Na to nesmíme zapomenout. Vždy nás však limituje čas.

Metody ***StartTransaction, Commit*** a ***Rollback*** jsou konfrontovány s databází ihned. Totéž platí pro modifikované záznamy. Pokud se tedy rozhodneme pro modifikaci většího počtu záznamů v rámci transakce, tak musíme vědět, že každý modifikovaný záznam je po úpravě ihned odeslán do databáze a v rámci příslušné tabulky uzamčen. To může být pro ostatní uživatele značně nemilé. Uživatel, který se pokusí modifikovat některý ze zamčených záznamů, je zablokován. Minimálně do doby, než je kolizní transakce dokončena nebo do vypršení expirace. Tyká se to samozřejmě pouze ovlivňovaných záznamů. Toto ruční řízení transakcí není pro většinu aplikací příliš vhodné.

Lepším způsobem je ovládání transakcí přímo klientem. Na pozadí. Programátor aplikace ví, které záznamy mají být izolovány či vykonány nepřerušitelně. Klient pak sám rozhodne, která operace je atomická a postará se o svázání záznamů v rámci jedné transakce. Pokud bychom například chtěli v rámci jedné transakce modifikovat dva záznamy v tabulce ***av***, provedli bychom to takto:

```
1 MyConnection1.StartTransaction;
2 Myquery2.Close;
3 Myquery2.SQL.Clear;
4 Myquery2.sql.Add('UPDATE audio_video.av SET AV_prijmeni = „Novák“ \
    WHERE AV_prijmeni = „Gott“ ');
5 Myquery2.sql.Add('UPDATE audio_video.av SET AV_prijmeni = „Jonesová“ \
    WHERE AV_prijmeni = „Jones“ ');
6 try
7   Myquery2.Execute;
8   MyConnection1.Commit;
9 Except
10  MyConnection1.Rollback;
11  Showmessage(‘Příkazy nebyly vykonány’);
12 End
```

Nejprve spustíme transakci. Potom sestavíme modifikační příkazy, které chceme provést atomicky. Poté je vykonáme v chráněném módu. Pokud dojde k jakékoliv chybě, je transakce odvolána v těle chráněného módu except. Pokud vše proběhne hladce, je zavolán příkaz commit. I tento příkaz podléhá chráněnému módu. Místo ***MyQuery2*** lze použít i ***MyCommand1***.

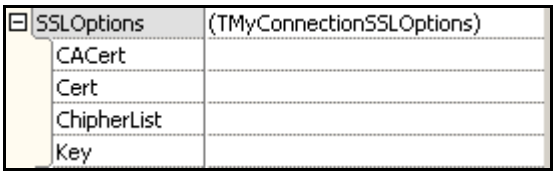
Transakci lze provádět také na úrovni cache klienta. Uživatel manipuluje se záznamy v cache paměti a databáze o tom neví. Až si uživatel řekne, že je to vše, tak pošle všechny změny do databáze hromadně. Jenom před ně umístí „start transaction“ a na konci „commit“. Výhoda této metody, které se říka ***CachedUpdates,*** je v tom, že transakce proběhne najednou a velice rychle. Nedojde tedy k blokování jiných uživatelů. Má to však jeden háček. Pošle-li do databáze uživatel *Abc* takovou to transakci, tak musí počítat s tím, že jiný uživatel *Xyz* mohl v tu samou chvíli poslat do databáze svoji transakci, ve které je několik stejných záznamů, jako má uživatel *Abc*. Výsledek bude záležet na stupni izolace. Jakákoliv jiná izolace, než ***serializable,*** bude znamenat veliké problémy. Může dojít k souběhu nebo uváznutí. Obě transakce se totiž budou vykonávat souběžně. V nejhorším případě bude některá transakce zlikvidována. Zamezit se tomu dá pouze tak, že izolaci nastavíme jako ***serializable***. Pak se každá transakce bude vykonávat samostatně.

K čemu je nám tedy ***CachedUpdates*** vůbec dobrý. Odpověď je snadná. Existují například záznamy, nad kterými musíme přemýšlet, nebo s nimi dlouho pracovat. Pak je není nutné permanentně blokovat. Podržíme si takový záznam v cache a modifikujeme ho třeba hodinu. Potom ho odešleme. Na druhou stranu tím riskujeme, že se nám záznam za tu hodinu diametrálně změnil. Opět máme dilema. Nebylo by lepší zamknout tento záznam? Univerzální odpověď neexistuje. Záleží to na datech a zase na datech. Neexistuje algoritmus, který by vyhověl všem. Je to o kompromisu.

Transakce v rámci našeho klienta je tedy možné aplikovat různými způsoby. Zásahy nejsou ani tak složité, jako spíš závisí na rozhodnutí programátora, jak je aplikovat na databázi. Záleží na mnoha aspektech. Například na povaze dat, struktuře databáze, rychlosti sítě a databázového stroje atd. Právě vybalancování všech požadavků určuje výkonnost a uživatelskou spokojenost hotové aplikace. Většina zmiňovaných aspektů je však navzájem v kolizi. Pokud vylepšíme jeden parametr, dva další se mohou zhoršit. Proto nikdy nelze vyhovět všem. Neexistuje totiž rozumný test. Databáze se může chovat vzorně při malém počtu záznamů. Problémy se začnou vynořovat až v okamžiku, kdy bude v tabulkách ohromné množství dat a budou k nim přistupovat tisíce uživatelů. Vlastní náprava pak probíhá až za běhu aplikace. Což může být problém.

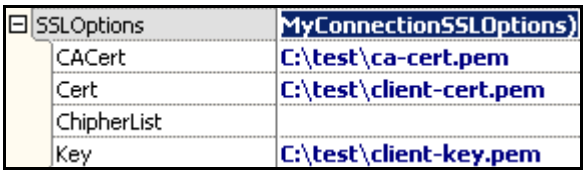
3.10 Bezpečné připojení

Nyní si ukážeme jak naprogramovat SSL zabezpečení. Je to velice snadné. Stačí modifikovat 3 parametry komponenty ***MyConnection1*** (Obr. 74).



Obr. 74. Delphi - aktivace SSL.

Jedná se o položky pro vložení ověřovacího certifikátu, privátního a soukromého klíče klienta. ***ChipherList*** umožňuje specifikovat seznam přípustných šifer. V kapitole o SSL jsme se naučili generovat certifikáty. Uložili jsme si je do adresáře ***c:\test***. Stačí je vložit do připravených řádků (Obr. 75).



Obr. 75. Delphi - vložené certifikáty SSL.

Nyní stačí spustit démona mysqld s parametry pro komunikaci SSL. Nastavit požadavek SSL komunikace u univerzálního klienta připojení a zkompilovat aplikaci. Komunikace bude probíhat šifrovaně. Jedná se o velmi elegantní a jednoduché řešení. My tuto funkci do klienta nepřidáme. Jednak ji nepotřebujeme a také bychom porušili podmínku jednosouborové aplikace. Certifikáty jsou další soubory navíc a to je pro nás nepřípustné. Zvídavému čtenáři však nic nebrání, aby si tuto funkci MyDAC vyzkoušel sám.

ZÁVĚR

Prvním cílem této diplomové práce bylo více porozumět databázi MySQL. Popsali jsme si autentizaci i autorizaci při navazování spojení s databází. Ponořili jsme se dostatečně hluboko do transakcí a jejich izolací. Dokázali jsme zabezpečit navazované spojení a také jsme uvedli postupy pro zvyšování výkonu databáze.

Druhým cílem pak byla úprava klienta z bakalářské práce, jehož vlastnosti jsme podstatně vylepšili. Nyní umí přihlašovat a registrovat uživatele. Měnit jejich oprávnění. Automaticky rušit staré neaktivní účty. Po úpravě se dokáže rovněž vypořádat se svými paralelně spuštěnými kopiemi. Je tedy schopen víceuživatelského provozu.

Diplomová práce poskytuje ideální prostor pro její další rozšíření. Tím je návrh a optimalizace databázové struktury.

CONCLUSION

The first objective of this diploma thesis was to understand more to the MySQL database. We have described the authentication as well as authorisation while establishing contact with the database. We have got into the transactions and their isolations in enough details. We have managed to ensure the contact and also we have stated a procedure of the database output increase.

The second objective was the client's adjustment that was from the bachelor thesis and whose features have been significantly improved. Now he can sign on and register users, change their competences, cancel old non-active accounts automatically. After adjustment, he is able to cope with his concurrently turned-on copies. Hence, he is capable of a multiuser operation.

The diploma thesis provides an ideal situation for its further extent. That is a proposal and the database stucture optimalisation.

SEZNAM POUŽITÉ LITERATURY

Monografie:

[1] STEPHENS, Ryan; PLEW, Ron; JONES, Arie D. *Naučte se SQL za 28 dní* : Computer Press, únor 2010. 728 s. ISBN 978-80-251-2700-1.

[2] GILMORE, W. Jason. *VELKÁ KNIHA PHP a MYSQL 5 :Kompendium znalostí pro začátečníky i profesionály.* 2. přeprac. vyd.: Zoner Press, 2005. 864s. ISBN 80-86815-53-6

[3] KOFLER, Michael . *Mistrovství v MySQL 5: Kompletní průvodce webového vývojáře.* 1. vyd. Brno : Computer press, 2007. 808 s. ISBN 978-80-251-1502-2

[4] OPPEL, Andy. *SQL bez předchozích znalostí.* Dotisk.: Computer Press, říjen 2008. 260 s. ISBN 978-80-251-1707-1.

[5] NORTHCUTT, Stephen, et al. *Bezpečnost počítačových sítí : Kompletní průvodce návrhem, implementací a údržbou zabezpečené sítě*: Computer Press, říjen 2005. 592 s. ISBN 80-251-0697-7.

[6] SVOBODA, Luděk, et al. *1001 tipů a triků pro Delphi : 2. aktualizované vydání.* Brno : Computer Press, 2003. 546 s. ISBN 80-7226-488-5.

[7] SWAN, Tom. *Mistrovství v delphi 4: Kompletní průvodce pro tvorbu aplikací.* Pavel Machek, David Hanousek, Luděk Hořčička. 1. vyd. Praha : Computer Press, 1999. 830 s. Programování. ISBN 80-7226-173-8.

[8] ZEMAN, Eduard. *Jednosouborový databázový klient. : Databáze typu klient-server.* 2009. 65 s. Bakalářská práce. UTB, Inženýrská informatika. Vedoucí bakalářské práce RNDr. Ing. Miloš Krčmář

Internetové zdroje:

[9] KADLEC, Josef. *Linuxsoft.cz* [online]. 2004 [cit. 2011-04-21]. Linux v příkazech - OpenSSL. Dostupné z WWW: <http://www.linuxsoft.cz/article.php?id_article=389>.

[10] SEIDLER, Kai 'Oswald' . *Apache friends - xampp* [online]. 2011, Thu Jan 27 2011 [cit. 2011-04-28]. XAMPP. Dostupné z WWW: <http://www.apachefriends.org/en/xampp-windows.html >.

[11] *Data Access Components for MySQL Overview* [online]. c1998-2009 [cit. 2009-04-12]. Dostupný z WWW: < http://www.devart.com/mydac/download.html>.

[12] *MySQL* [online]. MySQL AB, Sun Microsystems, Inc., c1995-2009 [cit. 2011-04-12]. Dostupný z WWW: <http://www.mysql.com/downloads/mysql/>

[13] Hlavn%C3%AD strana. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 14. 11. 2002, last modified on 5. 1. 2011 [cit. 2011-04-29]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Hlavn%C3%AD_strana>.

[14] *MySQL* [online]. MySQL AB, Sun Microsystems, Inc., c1995-2009 [cit. 2011-04-12]. Dostupný z WWW: < http://dev.mysql.com/doc/>

[15] *MySQL* [online]. MySQL AB, Sun Microsystems, Inc., c1995-2009 [cit. 2011-04-12]. Dostupný z WWW: <http://dev.mysql.com/downloads/gui-tools/5.0.html

[16] *MySQL* [online]. MySQL AB, Sun Microsystems, Inc., c1995-2009 [cit. 2011-04-12]. Dostupný z WWW: < http://dev.mysql.com/downloads/workbench/>

[17] *Embarcadero Technologies* [online]. 2011 [cit. 2011-05-09]. Database Software and Developer Tools. Dostupné z WWW: <http://www.embarcadero.com/products>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

Apache	Webový server.
CA	CERTIFICATE AUTHORITY - certifikační autorita.
CPU	CENTRAL PROCESSING UNIT - centrální procesor počítače.
HW	HARDWARE
InnoDB	Typ databázové tabulky podporující transakce.
IP	INTERNET PROTOCOL - identifikace síťového rozhraní.
MyDAC	Program pro komunikaci klienta a databáze.
MyISAM	Typ databázové tabulky.
MySQL	Databázový program.
PC	PERSONAL COMPUTER - osobní počítač.
PHP	PERSONAL HOME PAGE - hypertextový preprocesor. Jedná se o skriptovací jazyk na straně serveru.
RAM	RANDOM ACCESS MEMORY - paměť s libovolným přístupem.
RO	READ ONLY - pouze pro čtení.
SQL	STRUCTURED QUERY LANGUAGE - databázový jazyk.
SSD	SOLID STATE DRIVE - pevné disky bez pohyblivých částí.
SSL	SECURE SOCKETS LAYER - zabezpečující vrstva přenosového kanálu.
SW	SOFTWARE
TCP/IP	TRANSMISSION CONTROL PROTOCOL / INTERNET PROTOCOL - síťový protokol
VPN	VIRTUAL PRIVATE NETWORK - virtuální privátní síť.
wikipedia	Internetová otevřená encyklopedie.

SEZNAM OBRÁZKŮ

Obr. 1. XAMPP - Control Panel Application. 14
Obr. 2. XAMPP - uvítací stránka. 14
Obr. 3. PhpMyAdmin – seznam databází. 15
Obr. 4. Mysql - příkazový řádek programu. 16
Obr. 6. PhpMyAdmin - přehled struktury databáze mysql. 17
Obr. 7. PhpMyAdmin - část struktury tabulky user. 17
Obr. 8. PhpMyAdmin - detail obálky. 18
Obr. 9. PhpMyAdmin - data tabulky user. 18
Obr. 10. PhpMyAdmin - modifikační formulář. 19
Obr. 11. Mysql - chyba v přihlášení. 19
Obr. 12. PhpMyAdmin - úvodní stránka a reset privilegii. 20
Obr. 13. PhpMyAdmin - detail tabulky user. 20
Obr. 14. PhpMyAdmin - aktivace funkce PASSWORD při modifikaci hesla. 21
Obr. 15. PhpMyAdmin - zašifrované heslo. 21
Obr. 16. Mysql - změna hesla. 22
Obr. 17. Mysql - přehled databází, nastavení pracovní databáze a výpis záznamů. 23
Obr. 18. PhpMyAdmin - vytvoření uživatele Jan. 25
Obr. 19. PhpMyAdmin – vytvoření obecného uživatele. 25
Obr. 20. Mysql - vytvoření uživatele Dagmar. 25
Obr. 21. Mysql - vytvoření uživatele Eda. 26
Obr. 22. PhpMyAdmin - seznam uživatelů a jejich globální privilegia. 26
Obr. 23. Mysql - autentizace uživatele Dagmar. 27
Obr. 24. Mysql - pokus o vstup do databáze mysql. 28
Obr. 25. Mysql - založení nové databáze. 28
Obr. 26. PhpMyAdmin - výsek tabulky db. 29
Obr. 26. PhpMyAdmin - oprávnění pro uživatele Dagmar v tabulce db. 29
Obr. 27. Mysql - založení nové tabulky transakce. 30
Obr. 28. PhpMyAdmin - struktura tabulky transakce. 30
Obr. 29. Mysql - pokus o smazání tabulky transakce. 31
Obr. 30. Mysql - výpis tabulky transakce. 35
Obr. 31. Mysql - tabulka transakce po havárii. 36

Obr. 32. Mysql – nedokončená transakce. 37
Obr. 33. Mysql - dokončená transakce příkazem COMMIT. 38
Obr. 34. Mysql - odvolaná transakce příkazem ROLLBACK. 39
Obr. 35. Mysql - supertransakce. 40
Obr. 35. Mysql - připojení dvou uživatelů na tabulku transakce. 43
Obr. 37. Mysql - SELECT za běhu transakce. 43
Obr. 38. Mysql - blokováni dotazu SELECT. 44
Obr. 39. Mysql - timeout pro dotaz SELECT. 44
Obr. 40. Mysql - úspěšné vykonání transakce i dotazu SELECT. 44
Obr. 41. Mysql - izolace klientů. 46
Obr. 42. Mysql - nestandardní chování transakce. 47
Obr. 43. Mysql - funkční transakce s blokováním na úrovni záznamu. 47
Obr. 44. Mysql - ukázka uváznutí (deadlock). 49
Obr. 45. Mysql - dotaz na SSL funkcionalitu. 52
Obr. 46. Mysql - povolena komunikace SSL. 52
Obr. 47. OpenSSL - certifikáty a klíče. 54
Obr. 48. Mysql - Démon mysqld s podporou ssl. 54
Obr. 49. Mysql - přihlášení uživatele Jan bez SSL. 55
Obr. 50. Mysql - uživatel Eda bez SSL. 55
Obr. 51. Mysql - funkční SSL spojení nejnižší úrovně. 55
Obr. 52. Mysql - výpis SSL proměnných serveru MySQL. 56
Obr. 53. Mysqld - fragment konfiguračních direktiv MySQL. 58
Obr. 54. Windows XP - správce úloh. 60
Obr. 55. MySQL Administrátor. 61
Obr. 56. PhpMyAdmin - výpis tabulky user. 68
Obr. 57. PhpMyAdmin - výpis tabulky db. 68
Obr. 58. Mysql – registrace pomocí MySQL. 69
Obr. 59. PhpMyAdmin - aktualizovaný výpis tabulky db. 69
Obr. 60. Mysql - nastavení univerzálního uživatele. 72
Obr. 61. Delphi 2007 – ukázka pracovní plochy programu. 73
Obr. 62. Delphi - modifikace připojení 74
Obr. 63. Klient – úprava. Vlevo před změnou, vpravo po změně. 75

Obr. 64. PhpMyAdmin - modifikovaná tabulka cfg. 75
Obr. 65. Delphi - vložení nové jednotky. 76
Obr. 66. Delphi - přihlašovací formulář. 76
Obr. 67. Delphi - ukázka kódu přihlašovacího formuláře. 77
Obr. 68. Klient – přihlášení. 78
Obr. 69. Klient – chyba připojení. 81
Obr. 70. Klient – odchycení chyby. 81
Obr. 71. Klient – česká lokalizace. 82
Obr. 72. Klient – potvrzení modifikace. 83
Obr. 73. Delphi - nastavení izolace transakce. 84
Obr. 74. Delphi - aktivace SSL. 87
Obr. 75. Delphi - vložené certifikáty SSL. 88

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA PI: CD disk obsahující:

- Diplomovou práci
- Zdrojové kódy projektu
- Zkompilovanou aplikaci

PŘÍLOHA PII: Pracovní plocha klienta.

PŘÍLOHA PIII: Tisková plocha klienta.

PŘÍLOHA P II: PRACOVNÍ PLOCHA KLIENTA

PŘÍLOHA P III: TISKOVÁ PLOCHA KLIENTA.