8

MySQL

Základní správu dat v otevřených souborech a v jednoduché, ale rychlé databázi `dbm` jsme již zvládli, takže se můžeme pustit do vyššího, plnohodnotného nástroje pro práci s daty, a sice do relačního databázového systému (někdy se mu také komplikovaněji říká relační systém řízení báze dat) neboli RDBMS (Relational Database Management System).

Dvěma nejznámějšími relačními databázovými systémy typu open source jsou zřejmě PostgreSQL a MySQL, i když jich existuje mnohem více. Existuje také řada komerčních relačních databází, jako je Oracle, Sybase a DB2, z nichž všechny mají velmi bohaté funkce a pracují na mnoha platformách. Oblíbenou alternativou na komerčním trhu je také Microsoft SQL Server, který je ovšem omezen jen na platformu Windows. Každý z těchto balíků má své silné stránky; my se ale v této knížce budeme věnovat výhradně databázi MySQL, a to jednak z důvodů úspory místa, jednak proto, že celá kniha hovoří o otevřeném softwaru open source.

MySQL se zrodilo někdy okolo roku 1984, ale dnes již několik roků probíhá jeho vývoj a údržba komerčně, pod hlavičkou společnosti MySQL AB. Protože databáze MySQL je typu open source, zaměňují se jeho prvky často s jinými projekty open source. Stojí za zmínku, že i když můžeme databázi MySQL v mnoha situacích používat podle podmínek GNU General Public License (GPL), za jistých okolností si naopak k produktu musíme zakoupit komerční licenci. Přesné podmínky si vždy ověřte na webových stránkách `www.mysql.com` a rozhodněte, které vydání MySQL pro dané účely použijete.

Jestliže potřebujete databázi typu open source, podmínky MySQL pod GPL jsou pro vás nepřijatelné a komerční licenci se vám nechce kupovat, zkuste se podívat na PostgreSQL, jehož licenční podmínky jsou méně restriktivní (alespoň v době vzniku této knihy) a které tak může být velice vhodnou alternativou. Podrobné informace najdete na webových stránkách `www.postgresql.org`.

> Chcete-li se o databázi PostgreSQL naučit více, podívejte se buď do knihy *Beginning Databases with PostgreSQL: From Novice to Professional, Second Edition* (Apress, 2005, ISBN 1590594789), anebo v češtině do knihy *PostgreSQL Praktický průvodce* (Computer Press, 2003, ISBN: 80-722-6954-2).

Tato kapitola se věnuje následujícím tématům databáze MySQL:

- instalace MySQL,
- administrativní příkazy, nezbytné pro obsluhu MySQL,
- základní funkce MySQL,
- rozhraní API pro přístup k databázím MySQL z programů v jazyce C,
- vytvoření relační databáze, kterou můžeme v aplikaci pro sbírku CD využívat z jazyka C.

Instalace

Ať už máte na počítači jakoukoli variantu Linuxu, databázi MySQL v ní máte nejspíše předem sestavenou a připravenou k instalaci. Hotové balíky s instalací jsou tak například k dispozici v současných verzích distribuce Red Hat, SuSE i Ubuntu. Obecně doporučujeme využít právě tyto hotové balíky, protože tak databázi MySQL „rozchodíte“ nejsnáze a nejrychleji. Pokud v dané distribuci balík MySQL nemáte nebo pokud si chcete nainstalovat nejnovější verzi, můžete si stáhnout příslušné binární i zdrojové balíky z webových stránek MySQL.

V této kapitole budeme hovořit pouze o instalaci hotových verzí MySQL.

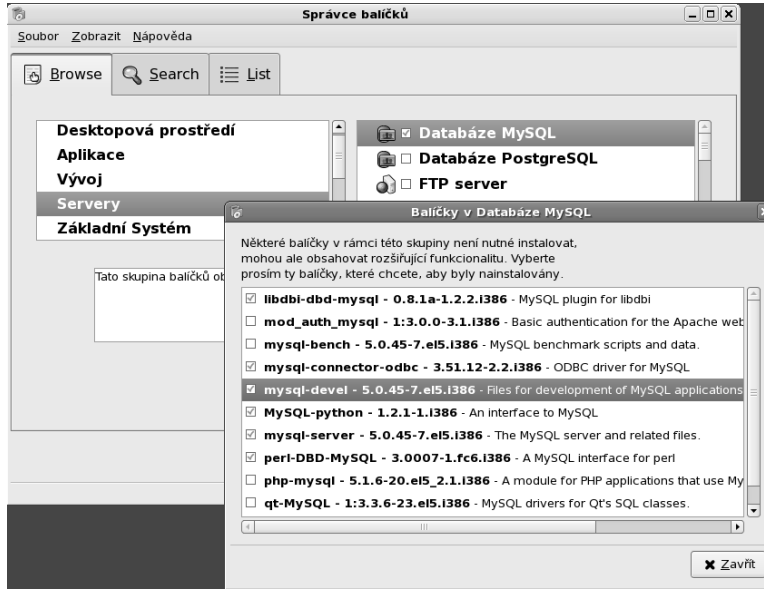
Balíky MySQL

Pokud se z jakéhokoli důvodu rozhodnete stáhnout novou verzi MySQL a nepoužít hotovou, budeme pro účely této knížky předpokládat komunitní vydání ve Standardním sestavení. K dispozici jsou také balíky Max a Debug. Z toho balíky Max obsahují různé doplňující funkce, jako je podpora méně obvyklých typů souborů pro záznam dat, a různé pokročilé funkce, jako je clustering. Balíky typu Debug jsou kompilovány s dalším kódem a informacemi potřebnými pro ladění; doufejme, že tyto informace nízké úrovně nebudete nikdy potřebovat.

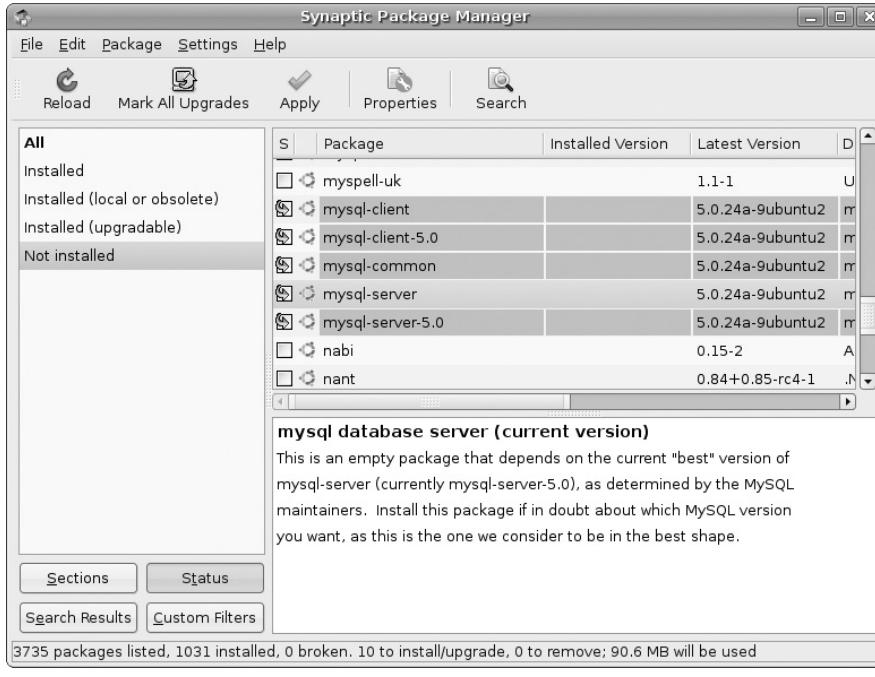
> Ladicí verze Debug nepoužívejte v produkčním prostředí; z důvodu podpory zmíněných ladicích funkcí jsou pomalejší.

Pro vývoj aplikací MySQL si musíte kromě vlastního serveru nainstalovat i vývojové knihovny. Obvykle najdete přímo ve správci balíků zvláštní volbu MySQL; stačí tedy zkontrolovat, jestli jsou vývojové knihovny nainstalovány také. Na obrázku 8.1 vidíme správce balíků z distribuce Fedora, který je připraven k instalaci MySQL i doplňujícího vývojového balíku.

V jiných distribucích je uspořádání balíků trochu jiné. Na obrázku 8.2 je například správce balíků Synaptic z distribuce Ubuntu, v němž je opět připravena instalace MySQL.



Obrázek 8.1



Obrázek 8.2

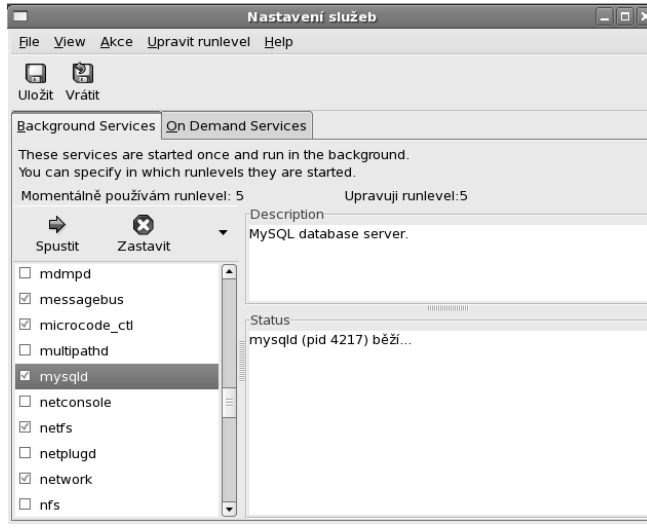
V rámci instalace MySQL se zároveň vytvoří uživatelský účet „mysql“, který je zároveň výchozím uživatelským jménem pro spuštění démonu MySQL serveru.

Po instalaci všech potřebných balíků musíme ověřit, jestli se MySQL spouští automaticky. V době vzniku těchto řádků platilo automatické spouštění jen v některých distribucích, jako například Ubuntu, zatímco jiné, třeba Fedora, to nedělaly. Zjistit, jestli MySQL běží, není naštěstí vůbec složité:

```
$ ps -el | grep mysqld
```

Uvidíme-li ve výpise alespoň jeden proces s názvem `mysqld`, znamená to, že server byl správně spuštěn. V mnoha systémech uvidíme také proces `safe_mysqld`, jehož úlohou je spuštění vlastního procesu `mysqld` pod správným uživatelským ID.

Potřebujete-li server MySQL spustit, restartovat nebo zastavit, můžete využít grafického ovládacího panelu služeb. Panel s názvem Service Configuration z distribuce Fedora je na obrázku 8.3.



Obrázek 8.3

Z editoru pro konfiguraci služeb také zjistíme, jestli se server MySQL bude spouštět při každém startu systému Linux automaticky.

Konfigurace po instalaci

Pokud všechno proběhlo v pořádku, je MySQL nainstalováno a je spuštěno s výchozí množinou voleb. Správnost tohoto předpokladu si můžeme snadno ověřit:

```
$ mysql -u root mysql
```

Objeví-li se zpráva „Welcome to the MySQL monitor“ a po ní výzva `mysql>`, znamená to, že server správně běží. V tomto okamžiku se samozřejmě může k serveru připojit kdokoli a získá oprávnění administrátora, tomuto problému se ale budeme věnovat za chviličku. Zkuste napsat `\s` a zjistit tak o daném serveru další informace. Po skončení experimentů napište `quit` nebo `\q` a chod monitoru tak ukončete.

Podrobnější informace o serveru vyvoláte příkazem `mysql -?`. Ve výpisu si všimneme jedné důležité věci: za seznamem argumentů bývá obvykle něco jako `Default options are read from the following files in the given order:`. Z řádků za touto frází zjistíme, kde se nacházejí konfigurační soubory serveru MySQL; obvykle je to soubor `/etc/my.cnf`, i když v některých distribucích, například Ubuntu, se používá soubor `/etc/mysql/my.cnf`.

Stav běžícího serveru je možné zkontrolovat také příkazem `mysqladmin`:

```
$ mysqladmin -u root version
```

Z výsledků nejen ověříme, že je server správně spuštěný, ale také zjistíme číslo jeho verze.

Další velice užitečnou funkcí příkazu `mysqladmin` je ověření všech konfiguračních voleb běžícího serveru; k tomu slouží volba `variables`:

```
$ mysqladmin variables
```

Výsledkem příkazu bude dlouhý seznam přiřazených hodnot proměnných. Mezi nejužitečnější proměnné patří `datadir` (z té zjistíme, kam MySQL ukládá data) a `have_innodb` (ta obvykle obsahuje hodnotu `YES` a indikuje, jestli je podporováno úložiště typu InnoDB). MySQL podporuje totiž celou řadu typů úložišť (storage engines) neboli podkladových implementačních ovladačů pro záznam či ukládání dat. Nejběžnějšími a také nejužitečnějšími jsou InnoDB a MyISAM, existují ale i jiné, například paměťové úložiště Memory Engine, které vůbec nepoužívá trvalá záznamová zařízení, a úložiště CSV, které zapisuje do souborů s čárkami oddělenými hodnotami. Různá úložiště mají různé funkce a výkonnost. Pro obecné používání databází doporučujeme úložiště InnoDB, protože je rozumným kompromisem mezi výkonností a podporou vynucování relací mezi jednotlivými datovými prvky. Nemáte-li podporu úložiště InnoDB zapnutou, podívejte se do konfiguračního souboru `/etc/my.cnf`, na začátek řádku `skip-innodb` zapište znak komentáře # a pomocí editoru služeb restartujte MySQL. Pokud ani toto nepomůže, možná máte verzi MySQL, zkompilovanou bez podpory InnoDB. V takovém případě najdete verzi s podporou InnoDB na webových stránkách MySQL. Pro účely této kapitoly nevadí, pokud budete používat alternativní úložiště MyISAM, které je ostatně v mnoha distribucích výchozí.

Jakmile bezpečně víte, že je součástí binárního kódu serveru i podpora InnoDB, musíte ho ještě určit za výchozí úložiště v konfiguračním souboru `/etc/my.cnf`, jinak se jako výchozí použije opět MyISAM. Potřebné úpravy jsou velice jednoduché: do sekce `mysqld` přidejte řádek s textem `default-storage-engine=INNODB`. Začátek souboru by měl po této úpravě vypadat třeba takto:

```
[mysqld]
default-storage-engine=INNOD
datadir=/var/lib/mysql
...
```

Po zbytek kapitoly předpokládáme, že je za výchozí úložiště určeno právě InnoDB.

V produkčním prostředí potřebujeme často také změnit výchozí umístění dat, určené hodnotou proměnné `datadir`. I toto nastavení určíme v sekci `mysqld` konfiguračního souboru `/etc/my.cnf`. Pokud se například rozhodnete používat úložiště InnoDB, ukládat data na svazek `/vol02` a soubory protokolů na svazek `/vol03` a datovému souboru přiřadit počáteční velikost 10 MB s možností rozšíření, napíšeme do konfigurace řádky:

```
innodb_data_home_dir = /vol02/mysql/data
innodb_data_file_path = ibdata1:10M:autoextend
innodb_log_group_home_dir = /vol03/mysql/logs
```

Další podrobnosti a další možnosti konfigurace najdete v online manuálech na webových stránkách `www.mysql.com`.

> Pokud se server nespustí nebo pokud se po jeho spuštění nemůžete připojit k databázi, přečtěte si následující část o řešení problémů v instalaci.

Vzpomenete si ještě na tu bezpečnostní díru, o které jsme se zmínili o pár odstavců výš a díky které se mohl každý bez hesla přihlásit jako administrátor root? Po prvotní konfiguraci bychom měli databázový systém lépe zabezpečit. Nenechte se mýlit, že se tento uživatel v instalaci MySQL nazývá `root` – se superuživatelem `root` v Linuxu skutečně nemá nic společného a název `root` je v MySQL pouze výchozím označením administrátora. Ani uživatelská ID obou účtů z MySQL a z Linuxu nijak nesouvisí; MySQL má svoji vlastní, vestavěnou správu uživatelů a oprávnění. Podle výchozího nastavení se ovšem k serveru MySQL může přihlásit každý, kdo má v dané linuxové instalaci účet, a může se stát administrátorem. Jakmile tuto bezpečnostní díru odstraníme a oprávnění uživatele `root` v MySQL zpřísníme – například povolíme pod `root` přihlášení jen lokálnímu uživateli a nastavíme mu heslo – můžeme vytvořit i další uživatelské účty a oprávnění, které pro provoz konkrétní aplikace potřebujeme.

Heslo administrátora `root` lze nastavit několika způsoby a nejsnáze pak takto:

```
$ mysqladmin -u root password novéheslo
```

Takto bude počáteční heslo účtu `root` rovno `novéheslo`.

S touto metodou je ale spojen jeden problém, protože heslo zůstane v podobě čistého textu zapsáno v historii shellu, odkud si je kdokoli může zjistit, například při provádění našeho příkazu zadáním příkazu `ps`, anebo vyloženě obnovením z historie shellu. Vhodnější metodou je proto nastavit heslo z MySQL monitoru, a to příslušným příkazem SQL.

```
$ mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4

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

mysql> SET password=PASSWORD('tajnéheslo');
Query OK, 0 rows affected (0.00 sec)
```

Vyberte samozřejmě jiné heslo než náš „vtipný“ text `tajnéheslo` – heslo byste měli znát jen vy sami. Chcete-li někdy později heslo odstranit, stačí ve stejném příkazu zadat prázdné heslo.

> Všimněte si, že jsme příkaz SQL ukončili středníkem (`;`). Tento znak není přesněji řečeno součástí příkazu SQL, ale pouze klientskému programu MySQL říká, že již požadujeme jeho provedení. Klíčová slova jazyka SQL zapisujeme také velkými písmeny, například `SET`. To není nutné, protože syntaxe MySQL povoluje klíčová slova velkými i malými písmeny, my ale tuto konvenci používáme nejen v knížce, ale i v naší každodenní práci, protože kód SQL je takto lépe čitelný.

Nyní se podíváme na tabulku oprávnění a zkontrolujeme, jestli bylo heslo nastaveno. Nejprve se příkazem `use` přepneme na databázi `mysql` a poté provedeme dotaz do interních tabulek:

```
mysql> use mysql
mysql> SELECT user, host, password FROM user;
+------+-----------+------------------+
| user | host      | password         |
+------+-----------+------------------+
| root | localhost | 2dxf8e9c23age6ed |
| root | fc7blp4e  |                  |
|      | localhost |                  |
|      | fc7blp4e  |                  |
+------+-----------+------------------+
4 rows in set (0.01 sec)

mysql>
```

V předchozím příkazu jsme tedy vytvořili heslo uživatele `root` jen při připojení z počítače `localhost`. MySQL umí totiž ukládat různá oprávnění nejen pro různé uživatele, ale i pro různé třídy připojení, odvozené od hostitelského názvu. Dalším krokem v zabezpečení naší instalace bude tedy odstranění uvedených nežádoucích uživatelských účtů, které jsou součástí výchozí instalace MySQL. Takto z tabulky oprávnění odstraníme všechny uživatele kromě `root`:

```
mysql> DELETE FROM user WHERE user!= 'root';
Query OK, 2 rows affected (0.01 sec)
```

Druhým příkazem odstraníme jakékoli přihlášení uživatele `root` z jiného hostitele než `localhost`:

```
mysql> DELETE FROM user WHERE host!= 'localhost';
Query OK, 1 row affected (0.01 sec)
```

A konečně ve třetím příkazu zkontrolujeme, jestli v tabulce nejsou žádné nežádoucí uživatelské účty:

```
mysql> SELECT user, host, password FROM user;
+------+-----------+------------------+
| user | host      | password         |
+------+-----------+------------------+
| root | localhost | 2dxf8e9c23age6ed |
+------+-----------+------------------+
1 row in set (0.00 sec)

mysql>exit
```

Jak vidíme z výsledků, konečně máme v systému jen jediný přihlašovací účet `root`, který se navíc smí přihlásit jen z `localhost`.

A teď přichází hodina pravdy: dokážete se stále přihlásit pod heslem, které jste původně zadali? Tentokrát napíšeme parametr `-p`, který znamená, že MySQL musí vypsat výzvu k zadání hesla:

```
$ mysql -u root -p
Enter password:
```

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7

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

mysql>
```

Máme tedy běžící verzi MySQL, jejíž přístup je zablokován – k databázovému serveru se smí připojit jen uživatel `root` pod dříve zadaným heslem, a navíc jen z počítače `localhost`. Při připojení k MySQL můžeme heslo zadat také na příkazovém řádku, a to v parametru `--password`, například `--password=tajnéheslo`, nebo stručněji `-ptajnéheslo`; tento postup není ale příliš bezpečný, protože heslo je opět možné odhalit z příkazu `ps` nebo z historie shellu. Zápisu hesla na příkazovém řádku se ale někdy nevyhneme, především při připojení k MySQL ze skriptů.

Dalším krokem je vrácení potřebného uživatele či uživatelů. Stejně jako v celém linuxovém systému je i v MySQL krajně nevhodné provádět veškeré operace pod uživatelem `root` (pokud to není nezbytně nutné); vytvořte tedy účet běžného uživatele pro každodenní, normální používání.

Jak jsme si již řekli, uživatelé mohou mít různá oprávnění při připojení z různých počítačů; v našem příkladu má uživatel `root` z bezpečnostních důvodů povoleno jen přihlášení z lokálního počítače. Pro potřeby této kapitoly vytvoříme nyní nového uživatele `rick`, který bude mít poměrně široká oprávnění. Konkrétně se bude moci připojit těmito třemi způsoby:

- může se připojit z lokálního hostitele,
- může se připojit z libovolného počítače s IP adresou v intervalu od 192.168.0.0 do 192.168.0.255,
- může se připojit z libovolného počítače v doméně `wiley.com`.

Nejsnáze tohoto cíle dosáhneme bezpečným způsobem tak, že vytvoříme tři různé uživatele se třemi různými cestami připojení. Každý z „uživatelů“ může mít dokonce jiné heslo, které tak platí jen pro tu kterou síťovou adresu.

K vytvoření uživatelů a udělování oprávnění slouží příkaz `grant`, pomocí něhož nadefinujeme tři zmíněné cesty připojení. Parametr `IDENTIFIED BY` je poněkud zvláštním výrazem pro zadání počátečního hesla. Všimněte si zápisu apostrofů; ty uvádějte přesně jako v příkladu, jinak se uživatelé nevytvoří v souladu s našimi požadavky.

1. Vytvoříme lokální přihlášení uživatele `rick`:

   ```
   mysql> GRANT ALL ON *.* TO rick@localhost IDENTIFIED BY 'tajnéheslo';
   Query OK, 0 rows affected (0.03 sec)
   ```

2. Poté vytvoříme přihlášení, s nímž bude možné se připojit z libovolného systému v podsíti třídy C o adrese 192.168.0. Nezapomeňte opět zapsat apostrofy, které zde uzavírají interval povolených adres IP adres definovaných maskou `/255.255.255.0`:

   ```
   mysql> GRANT ALL ON *.* TO rick@'192.168.0.0/255.255.255.0'
          IDENTIFIED BY 'tajnéheslo';
   Query OK, 0 rows affected (0.00 sec)
   ```

3. Nakonec vytvoříme přihlášení, jehož prostřednictvím se `rick` bude moci připojit z libovolného počítače v doméně `wiley.com` (a opět nezapomeneme zapsat apostrofy):

```
mysql> GRANT ALL ON *.* TO rick@'%.wiley.com' IDENTIFIED BY 'tajnéheslo';
Query OK, 0 rows affected (0.00 sec)
```

4. A nyní se znovu podíváme do tabulky uživatelů a zkontrolujeme, jestli se všechny položky vytvořily správně:

```
mysql> SELECT user, host, password FROM mysql.user;
+------+---------------------------+------------------+
| user | host                      | password         |
+------+---------------------------+------------------+
| root | localhost                 | 2dxf8e8c17ade6ed |
| rick | localhost                 | 3742g6348q8378d9 |
| rick | %.wiley.com               | 3742g6348q8378d9 |
| rick | 192.168.0.0/255.255.255.0 | 3742g6348q8378d9 |
+------+---------------------------+------------------+
4 rows in set (0.00 sec)

mysql>
```

Výše uvedené příkazy i hesla musíte přirozeně změnit podle podmínek lokální konfigurace. Všimněte si zejména příkazu `GRANT ALL ON *.*`, který, jak jistě správně tušíte, dává uživateli `rick` neobyčejně silná oprávnění. Pro zkušeného uživatele je to jistě v pořádku, ale pro běžného uživatele je to rozhodně nevhodné. O příkazu `grant` budeme v této kapitole hovořit v části „Vytváření uživatelů a udělování oprávnění“, kde si mimo jiné ukážeme právě vytvoření běžného uživatele s omezenými oprávněními.

Tím by MySQL mělo být správně nainstalováno a spuštěno (není-li tomu tak, podívejte se do následující části textu), instalace by měla být zabezpečena a měl by v ní být běžný uživatel různý od `root`. Podíváme se tedy na řešení případných problémů po instalaci a poté se rychle vrátíme zpět k základům administrace databází MySQL.

Řešení problémů po instalaci

Jestliže připojení po zadání příkazu `mysql` selhalo, zkontrolujte nejprve systémovým příkazem `ps`, jestli proces serveru správně běží. Pokud jej ve výpise nenajdete, zkuste zadat příkaz `mysql_safed -log`. Ten zapíše do adresáře protokolů MySQL soubor s jistými doplňujícími informacemi. Můžete se také pokusit o přímé spuštění procesu `mysqld`; úplný seznam voleb příkazového řádku vyvoláte příkazem `mysqld --verbose --help`.

Je také možné, že sice server správně běží, ale že jednoduše odmítnul naše připojení. V takovém případě musíme jako další věc zkontrolovat přítomnost databází, zejména pak výchozí databáze MySQL s oprávněními. V distribucích Red Hat se běžně používá výchozí umístění `/var/lib/mysql`, v jiných distribucích to ale může být i jinde. Podívejte se proto do spouštěcího skriptu MySQL (například v adresáři `/etc/init.d`) a do konfiguračního souboru `/etc/my.cnf`. Další možnost je vyvolat program přímo, a to příkazem `mysqld --verbose`, a hledat zde proměnnou `datadir`. Po nalezení adresáře s databází zkontrolujeme, jestli je v ní přítomna přinejmenším výchozí databáze oprávnění (s názvem `mysql`) a že démon serveru toto umístění skutečně používá (je zadáno v souboru `my.cnf`).

Pokud se ani po tomto pokusu nepodaří připojit, zkuste pomocí editoru služeb zastavit server, dále zkontrolujte, jestli už neběží žádný proces `mysqld`, a poté jej restartujte a připojte se znovu. A pokud selže i tento pokus (jakkoli je to nepravděpodobné), můžete ještě sáhnout k poslední me-

todě, a sice odinstalovat MySQL a nainstalovat je celé znovu. Velmi užitečná je také dokumentace MySQL, kterou najdete na webových stránkách MySQL (je vždy aktuálnější než lokální manuálové stránky a navíc v ní najdete různé rady od uživatelů a diskusní fórum) a ve které se můžete podívat na řešení méně obvyklých problémů.

Administrace MySQL

Administraci MySQL usnadňuje několik pomocných programů, které jsou součástí jeho distribuce. Nejběžněji používaným je přitom program `mysqladmin`, kterému se spolu s jinými utilitami věnujeme na následujících řádcích.

Příkazy

Všechny příkazy MySQL s výjimkou příkazu `mysqlshow` přebírají nejméně tyto tři standardní parametry:

Volba příkazu	Parametr	Popis
`-u`	`uživjméno`	Jako výchozí hodnotu se všechny utility MySQL pokoušejí použít `uživjméno` MySQL shodné s aktuálním uživatelským jménem systému Linux. V parametru `-u` můžeme nicméně zadat jiné uživatelské jméno.
`-p`	`[heslo]`	Zadáme-li volbu `-p` bez řetězce hesla, vypíše se výzva k jeho zadání. Není-li parametr `-p` zadán vůbec, předpokládají příkazy MySQL, že žádné heslo není potřeba.
`-h`	`hostitel`	Slouží pro připojení k serveru na jiném hostiteli (u lokálního serveru je možné vždy vynechat).

> Ještě jednou doporučujeme rozhodně nezapisovat heslo na příkazový řádek, protože takto je viditelné ve výpisu `ps`.

Utilita myisamchk

Posláním utility `myisamchk` je kontrolovat a opravovat datové tabulky ve výchozím formátu tabulek MyISAM, který MySQL nativně podporuje, Normálně je třeba spouštět tuto utilitu pod uživatelským účtem `mysql`, vytvořeným při instalaci, a to z adresáře, v němž jsou tabulky umístěny. Při kontrole databáze tedy zadejte příkaz `su mysql`, přejděte do adresáře, jehož název je shodný s názvem databáze, a poté vyvolejte příkaz `myisamchk` s jednou nebo více volbami podle následující tabulky. Například:

```
myisamchk -e -r *.MYI
```

Nejběžněji používané volby příkazu `myisamchk` shrnuje stručná tabulka:

Volba příkazu	Popis
`-c`	Zkontrolovat chyby v tabulce.
`-e`	Provést rozšířené kontroly.
`-r`	Opravit nalezené chyby.

Podrobnější informace zjistíte v rozsáhlé nápovědě, kterou vyvoláte spuštěním příkazu `myisamchk` bez parametrů. Nad tabulkami typu InnoDB neprovádí utilita žádné operace.

Utilita mysql

Nejdůležitější a zároveň nejsilnější nástroj příkazového řádku v databázi MySQL, který umožňuje provádění téměř jakéhokoli úkonu na úrovni administrátora i běžného uživatele. Nástroj `mysql` můžete spustit z příkazového řádku, a pokud do něj zadáte v posledním parametru název databáze, nemusíte z monitoru zvlášť zadávat příkaz `use <databáze>`. Tímto příkazem spustíme například monitor pod uživatelem `rick`, požádáme o výzvu k zadání hesla (všimněte si mezery za volbou `-p`) a jako výchozí použijeme databázi `foo`:

```
$ mysql -u rick -p foo
```

Příkazem `mysql --help | less` vypíšeme po stránkách seznam dalších voleb příkazového řádku v monitoru `mysql`.

Spustíte-li MySQL bez zadání databáze, můžete konkrétní databázi vybrat zmíněným příkazem `use <názevdatabáze>` z MySQL, jak ostatně vidíte ze seznamu příkazů v následující tabulce.

Druhá možnost je spustit utilitu `mysql` v neinteraktivním režimu, příslušné příkazy zadat do vstupního souboru a ten nechat z příkazového řádku načíst. V takovém případě nezbývá, než zapsat heslo na příkazovém řádku:

```
$ mysql -u rick --password=tajnéheslo foo < sqlcommands.sql
```

Jakmile `mysql` přečte a zpracuje všechny příkazy zadané v souboru, vrátíme se zpět do výzvy příkazového řádku.

Pokud je klient `mysql` připojen k serveru, můžeme kromě standardní množiny SQL92 zadat také řadu podporovaných speciálních příkazů, které shrnuje následující tabulka:

Příkaz	Alternativní zkrácená forma	Popis
`help` nebo `?`	`\h` nebo `\?`	Vypíše seznam příkazů.
`edit`	`\e`	Editace příkazu. Konkrétní editor je určen z proměnné prostředí `$EDITOR`.
`exit` nebo `quit`	`\q`	Ukončí klienta MySQL.
`go`	`\g`	Spustí provedení příkazu.
`source <názevsouboru>`	`\.`	Provede příkazy SQL ze zadaného souboru.
`status`	`\s`	Vypíše stavové informace o serveru.
`system <příkaz>`	`\!`	Provede systémový příkaz.
`tee <názevsouboru>`	`\T`	Kopii veškerého výstupu připojí na konec souboru se zadaným názvem.
`use <databáze>`	`\u`	Použije určenou databázi.

Velmi důležitým příkazem z této tabulky je příkaz `use`. Server (démon) `mysqld` podporuje totiž práci s mnoha různými databázemi, které ale obsluhuje a spravuje jediný serverový proces. U mnoha

jiných databázových serverů, jako je Oracle nebo Sybase, se používá pojem *schéma*, zatímco v MySQL se setkáme s výrazem *databáze*. (Nástroj Query Browser v MySQL zná ale naopak *schéma*.) Každá databáze – ve smyslu databáze MySQL – je z velké části nezávislá množina tabulek. To znamená, že pro různé účely můžeme snadno vytvořit mnoho různých databází a k těmto různým databázím přiřadit různé uživatele; jejich efektivní správu bude přesto zajišťovat jeden stejný databázový server. Příkazem `use` se pak (za předpokladu příslušných oprávnění) mezi těmito databázemi přepínáme.

Speciální databáze *mysql*, vytvořená automaticky v každé instalaci MySQL, slouží jako hlavní úložiště dat, jako jsou uživatelé a oprávnění.

> SQL92 je nejčastěji používanou verzí normy ANSI SQL. Jejím úkolem je vytvořit jednotnou podobu databází SQL a umožnit vzájemnou spolupráci a komunikaci i mezi různými databázovými produkty.

Utilita mysqladmin

Tato utilita je nejdůležitějším nástrojem pro rychlou administraci databází MySQL. Kromě obvyklých parametrů podporuje také následující hlavní příkazy:

Příkaz	Popis
`create <název_databáze>`	Vytvoří novou databázi.
`drop <název_databáze>`	Odstraní databázi.
`password <nové_heslo>`	Změní heslo (ukázka viz výše).
`ping`	Zkontroluje, jestli server běží (je „naživu“).
`reload`	Znovu načte tabulky s udělenými oprávněními.
`status`	Vypíše stav serveru.
`shutdown`	Ukončí chod serveru.
`variables`	Zobrazí proměnné, které kontrolují činnost MySQL, spolu s jejich aktuálními hodnotami.
`version`	Uvádí číslo verze serveru a dobu, po kterou je již spuštěný.

Při spuštění z příkazového řádku bez parametrů vypíše příkaz `mysqladmin` úplný seznam voleb; k takovému příkazu je vhodné dopsat `| less`.

Utilita mysqlbug

Při troše štěstí byste nemuseli tuto utilitu nikdy používat. Jak již název napovídá, vytvoří tento nástroj zprávu s chybami, která se e-mailem odešle týmu údržby MySQL. Před odesláním můžete vygenerovaný soubor upravit a doplnit do něj další informace, jež mohou být pro vývojáře užitečné.

Utilita mysqldump

Neobyčejně užitečná utilita, která umožňuje provádět výpis celé databáze nebo její části do jediného souboru, a to ve tvaru příkazů SQL, jež můžeme později znovu načíst jak do MySQL, tak i do jiné relační databáze SQL. Kromě standardních parametrů s uživatelským jménem a heslem přebírá také název databáze a tabulky. Možnosti nástroje dále rozšiřují následující volby:

Příkaz	Popis
`--add-drop-table`	Do výstupního souboru přidá příkazy SQL, nezbytné pro odstranění tabulek; teprve za nimi následují příkazy pro vytvoření nových tabulek.
`-e`	Bude se používat rozšířená syntaxe vkládání. Tato syntaxe není standardním SQL, ale při výpisu velkého množství dat se mohou později do MySQL načíst mnohem rychleji.
`-t`	Výpisu podléhají jen data v tabulkách, nikoli příkazy a informace nutné pro vytvoření tabulek.
`-d`	Výpisu podléhají jen struktury tabulek, nikoli vlastní data.

Při výchozím nastavení odesílá nástroj `mysqldump` výsledky na standardní výstup, který pak můžeme přesměrovat do souboru,

Utilita je šikovná například při převodu (migraci) dat nebo při rychlém zálohování a díky implementaci architektury klient/server v MySQL s ní můžeme dokonce realizovat vzdálené zálohování, pokud je klient `mysqldump` nainstalován na jiném počítači. Jako příklad si uvedeme příkaz, který se připojí jako `rick` a vypíše databázi `myplaydb`:

```
$ mysqldump -u rick -p myplaydb > myplaydb.dump
```

V našem systému měla tato databáze jen jedinou tabulku a výsledný soubor vypadal tudíž takto:

```
-- MySQL dump 10.11
--
-- Host: localhost    Database: myplaydb
-- ------------------------------------------------------
-- Server version     5.0.37

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table 'children'
--

DROP TABLE IF EXISTS 'children';
CREATE TABLE 'children' (
  'childno' int(11) NOT NULL auto_increment,
  'fname' varchar(30) default NULL,
```

```
  'age' int(11) default NULL,
  PRIMARY KEY ('childno')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table 'children'
--

LOCK TABLES 'children' WRITE;
/*!40000 ALTER TABLE 'children' DISABLE KEYS */;
INSERT INTO 'children' VALUES
(1,'Jenny',21),(2,'Andrew',17),(3,'Gavin',8),(4,'Duncan',6),(5,'Emma',4),
(6,'Alex',15),(7,'Adrian',9);
/*!40000 ALTER TABLE 'children' ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
-- Dump completed on 2007-06-22 20:11:48
```

Utilita mysqlimport

Nástroj `mysqlimport` slouží k hromadnému načítání neboli importu dat do tabulky. Takto můžeme ze vstupního souboru načíst velké množství dat v textovém formátu. Jedinými povinnými parametry jsou název souboru a název databáze; v cílové databázi načte příkaz data do tabulky, jejíž název je shodný s názvem vstupního souboru (bez přípony). Textový soubor musí mít při importu stejný počet sloupců jako cílová tabulka a také jejich datové typy musí být kompatibilní. Výchozím oddělovačem dat je tabulátor.

Z textového souboru je možné načítat také příkazy SQL k provedení; stačí spustit příkaz `mysql` a vstup u něj přesměrovat ze souboru, jak jsme si ukázali dříve.

Utilita mysqlshow

Tato utilita vypisuje stručné informace o instalaci MySQL a jeho podřízených databázích:

- Bez parametrů vypisuje všechny dostupné databáze.
- Zadáme-li v parametru databázi, vypíše všechny tabulky v databázi.
- Zadáme-li název databáze a tabulky, vypíše všechny sloupce z tabulky.
- A pokud zadáme název databáze, tabulky a sloupce, uvede detailní informace o sloupci.

Vytváření uživatelů a udělování oprávnění

Jako administrátor MySQL budete mít na starosti mimo jiné údržbu uživatelů, tedy přidávání a odstraňování uživatelů z MySQL a řízení jejich oprávnění. Od verze MySQL 3.22 se pro správu uživatelských oprávnění používají příkazy `grant` a `revoke`, které zadáváme z MySQL monitoru – zde jsou tyto operace jistě jednodušší než ruční editace tabulek s oprávněními ve starších verzích MySQL.

Příkaz grant

Příkaz `grant` zhruba (nikoli však úplně) odpovídá syntaxi SQL92. Obecný formát vypadá takto:

```
grant <oprávnění> on <objekt> to <uživatel>
  [identified by heslo-uživatele] [with grant option];
```

Takto je možné přidělit několik typů <*oprávnění*>, které shrnuje následující tabulka:

Hodnota	Popis
alter	Oprávnění měnit tabulky a indexy.
create	Vytvářet databáze a tabulky.
delete	Odstraňovat data z databáze.
drop	Odstraňovat databáze a tabulky.
index	Provádět správu indexů.
insert	Přidávat (vkládat) data do databáze.
lock tables	Definovat uzamykání souborů.
select	Načítat (vybírat) data.
update	Modifikovat (aktualizovat) data.
all	Všechna výše popsaná oprávnění.

Některé příkazy mají další volby; parametr `create view` dává například uživateli oprávnění k vytváření pohledů. Úplný seznam oprávnění je nejlepší najít v dokumentaci ke konkrétní verzi MySQL, protože se s každou novou verzí MySQL mění. Existují také speciální oprávnění pro administrátory, kterým se zde ale věnovat nebudeme.

Objekt, nad nímž budeme oprávnění udělovat, identifikujeme zápisem

```
názevdatabáze.názevtabulky
```

přičemž ve shodě s linuxovými „tradicemi“ znamená znak * cokoliv, takže `*.*` jsou všechny objekty ve všech databázích a `foo.*` všechny objekty v databázi `foo`.

Pokud již zadaný uživatel existuje, oprávnění se upraví podle požadovaných změn. Jestliže uživatel neexistuje, vytvoří se jako nový a přiřadí se mu nová oprávnění. Jak jsme si ukázali, uživatele je možné zadat s tím, že se má přihlásit z konkrétního hostitelského počítače; stejný zápis můžeme využít i při zadávání oprávnění a využijeme tak plnou flexibilitu schématu MySQL pro oprávnění.

V syntaxi jazyka SQL se jako zástupný symbol používá znak `%` a má tedy podobný význam jako * v shellovém prostředí. Pro každé udělované oprávnění můžeme samozřejmě napsat samostatný příkaz, ale pokud se například rozhodneme udělit uživateli `rick` oprávnění přístupu z libovolného počítače v doméně `wiley.com`, vyjádříme uživatele jako:

```
rick@'%.wiley.com'
```

Jakýkoli zápis znaku * je nutné uvést do apostrofů, které jej tak oddělují od literálového textu.

Při řízení přístupu je možné zadat také síťovou adresu ve tvaru IP adresa/maska podsítě (N.N.N.N/M.M.M.M).

Už v předchozím příkladu jsme viděli, že zápisem `rick@'192.168.0.0/255.255.255.0'` udělíme uživateli `rick` právo přístupu z libovolného počítače v lokální síti; tvar `rick@'192.168.0.1'` pak znamená, že je jeho přístup omezen jen na jedinou pracovní stanici, a naopak výraz `rick@'192.0.0.0/255.0.0.0'` rozšiřuje přístup na libovolný počítač v síti třídy A s číslem 192.

A ještě jeden příklad:

```
mysql> GRANT ALL ON foo.* TO rick@'%' IDENTIFIED BY 'bar';
```

Tento příkaz udělí uživateli `rick` při zadání počátečního hesla `bar` a při připojení z libovolného počítače úplná oprávnění nad databází `foo`.

Pokud v tomto příkazu databáze `foo` neexistuje, bude mít uživatel `rick` oprávnění ji vytvořit příkazem jazyka SQL, `create database`.

Klauzule `IDENTIFIED BY` je nepovinná, obecně je však vhodné přiřadit každému uživateli co nejdříve po vytvoření účtu i heslo.

Velký pozor si musíte dávat v jednom nešťastném případě, a sice když uživatelská jména, hostitelské názvy nebo názvy databází obsahují podtržení _ – tento znak se totiž v jazyce SQL shoduje s libovolným jednotlivým znakem, podobně jako % definuje shodu s řetězcem znaků. Pokud možno se tedy znaku podtržení v uživatelských jménech a názvech databází vyhýbejte.

Volbu `with grant option` používáme zpravidla jen pro vytvoření sekundárního administrátora; obecně ale znamená, že nově vytvořený uživatel může svá oprávnění propůjčovat i jiným uživatelům. S volbou `with grant option` zacházejte proto opatrně.

Příkaz revoke

Administrátor má přirozeně právo nejen udělovat, ale také odebírat oprávnění. K tomu slouží příkaz `revoke`:

```
revoke <oprávnění> on <objekt> from <uživatel>
```

Formát je z velké části podobný příkazu `grant`, například:

```
mysql> REVOKE INSERT ON foo.* FROM rick@'%';
```

Příkaz `revoke` neumí ale odstranit uživatele. Chcete-li nějaký uživatelský účet úplně smazat, nestačí jednoduše upravit jeho oprávnění, ale nejprve mu příkazem `revoke` zrušte všechna oprávnění a poté jej smažte z tabulky `user` v interní databázi `mysql`:

```
mysql> use mysql
mysql> DELETE FROM user WHERE user = "rick"
mysql> FLUSH PRIVILEGES;
```

Protože jsme zde neurčili hostitelský počítač, odstraníme tímto příkazem všechny instance daného uživatele MySQL (zde `rick`). Po dokončení se nezapomeňte příkazem `use` vrátit k původní databázi, jinak můžete neúmyslně pokračovat v práci nad interní databází MySQL.

Uvědomte si, že příkaz `delete` patří do jiné skupiny než příkazy `grant` a `revoke`. Je to běžný příkaz SQL a my jím odstraňujeme záznamy z „běžné" tabulky – tímto způsobem totiž MySQL obsluhuje oprávnění. Pro dosažení potřebných změn musíme skutečně přímo aktualizovat tabulky oprávnění MySQL (a proto jsme také nejprve napsali příkaz `use mysql`).

Po aktualizaci tabulek musíme provést příkaz `FLUSH PRIVILEGES`, kterým serveru MySQL nařídíme opětovné načtení tabulek oprávnění.

Hesla

Chcete-li zadat hesla stávajícím uživatelům, kteří je ještě nemají, nebo chcete-li svoje vlastní heslo či heslo jiného uživatele změnit, musíte se k serveru MySQL připojit jako `root` a přímo aktualizovat informace o uživateli. Například:

```
mysql> use mysql
mysql> SELECT host, user, password FROM user;
```

Dostaneme zhruba takovýto výsledek:

```
+-----------+----------+------------------+
| host      | user     | password         |
+-----------+----------+------------------+
| localhost | root     | 67457e226a1a15bd |
| localhost | foo      |                  |
+-----------+----------+------------------+
2 rows in set (0.00 sec)
```

Tímto příkazem můžeme uživateli `foo` přiřadit heslo `bar`:

```
mysql> UPDATE user SET password = password('bar') WHERE user = 'foo';
```

Pro kontrolu můžeme nyní znovu vypsat příslušné sloupce z tabulky `user`:

```
mysql> SELECT host, user, password FROM user;
+-----------+----------+------------------+
| host      | user     | password         |
+-----------+----------+------------------+
| localhost | root     | 65457e236g1a1wbq |
| localhost | foo      | 7c9e0a41222752fa |
+-----------+----------+------------------+
2 rows in set (0.00 sec)
mysql>
```

Vidíme, že uživatel `foo` již skutečně má heslo. Nezapomeňte se vrátit zpět k původní databázi.

Ve verzi MySQL 4.1 se schéma ukládání hesel oproti starším verzím změnilo. Z důvodu zpětné kompatibility je nicméně zachována i možnost změny hesla pomocí starého algoritmu, který v případě potřeby vyvoláme pomocí funkce `OLD_PASSWORD('nové heslo')`.

Vytvoření databáze

Naším dalším krokem bude vytvoření databáze. Dejme tomu, že se bude nazývat `rick` – a jak si jistě vzpomínáte, v systému je již uživatel stejného jména. Nejprve uživateli `rick` udělíme široká oprávnění, aby tak mohl především vytvářet nové databáze. Toto oprávnění je velice užitečné zejména na vývojovém systému, protože pro vývojáře znamená vysokou flexibilitu:

```
mysql> GRANT ALL ON *.* TO rick@localhost IDENTIFIED BY 'tajnéheslo'
```

Poté si správnost oprávnění vyzkoušíme – přihlásíme se jako `rick` a vytvoříme novou databázi:

```
$ mysql -u rick -p
Enter password:
...
mysql> CREATE DATABASE rick;
Query OK, 1 row affected (0.01 sec)
mysql>
```

Nyní požádáme MySQL o přepnutí pod novou databázi:

```
mysql> use rick
```

A dále již můžeme do databáze naplnit tabulky a všechny požadované informace. Při dalším přihlášení můžeme na konec příkazového řádku zadat název databáze a tím si „ušetřit“ zápis příkazu `use`:

```
$ mysql -u rick -p rick
```

Po zadání správného hesla (na výzvu) se automaticky ocitneme v databázi `rick`.

Datové typy

Server MySQL se tedy správně rozběhl, uživateli jsme nadefinovali bezpečné přihlášení a vytvořili jsme i databázi. Co dál? Nyní musíme v databázi vytvořit tabulky se sloupci, do nichž budeme zapisovat vlastní data. K tomu se ale nejprve musíme seznámit s datovými typy, které MySQL podporuje.

Datové typy v MySQL jsou poměrně standardní, takže se stručně podíváme jen na ty nejdůležitější. Jako vždy najdete detailní informace v manuálu na webových stránkách MySQL.

Logický typ

Sloupec logického neboli booleovského typu definujeme pomocí klíčového slova `BOOL`. Jak si jistě sami domyslíte, může obsahovat hodnoty `TRUE` a `FALSE` (pravda a nepravda); třetí hodnotou je speciální „neznámá“, `NULL`.

Znakové typy

MySQL nabízí několik znakových typů, které shrnuje následující tabulka. První tři jsou standardní, zbývající tři jsou pak speciální typy MySQL. Doporučujeme přitom se pokud možno držet standardních typů.

Definice	Popis
`CHAR`	Jednotlivý znak.
`CHAR(N)`	Znakový řetězec v délce přesně *N* znaků; kratší řetězec se zprava doplní mezerami. Maximální délka 255 znaků.
`VARCHAR(N)`	Pole proměnné délky s *N* znaky. Maximální délka 255 znaků.
`TINYTEXT`	Podobné jako `VARCHAR(N)`.
`MEDIUMTEXT`	Textový řetězec obsahující až 65 535 znaků.
`LONGTEXT`	Textový řetězec obsahující až $2^{32} - 1$ znaků.

Číselné typy

Všechny číselné typy můžeme dále rozdělit na celočíselné a na reálné typy s pohyblivou řádovou čárkou (podle následující tabulky):

Definice	Typ	Popis
TINYINT	Celočíselný	8bitový datový typ.
SMALLINT	Celočíselný	16bitový datový typ.
MEDIUMINT	Celočíselný	24bitový datový typ.
INT	Celočíselný	32bitový datový typ. Jedná se o standardní datový typ, vhodný pro obecné použití.
BIGINT	Celočíselný	64bitový datový typ se znaménkem.
FLOAT(P)	Reálný	Číslo s pohyblivou řádovou čárkou a nejméně *P* číslicemi přesnosti.
DOUBLE(D, N)	Reálný	Číslo s pohyblivou řádovou čárkou ve dvojnásobné přesnosti se znaménkem, s *D* číslicemi a *N* desetinnými místy.
NUMERIC(P, S)	Reálný	Reálné číslo s celkem *P* číslicemi, z toho *S* číslic za desetinnou čárkou. Na rozdíl od typu `DOUBLE` se jedná o přesné číslo, takže se hodí například pro ukládání peněžních údajů; na druhé straně se zpracovává méně efektivně.
DECIMAL(P, S)	Reálný	Synonymum pro `NUMERIC`.

Obecně doporučujeme držet se datových typů `INT`, `DOUBLE` a `NUMERIC`, protože se nejvíce blíží standardním datovým typům SQL. Ostatní typy jsou nestandardní a v jiných databázových systémech nemusí být k dispozici, takže následný převod databáze do jiného prostředí může znamenat problémy.

Časové typy

K dispozici máme také pět datových typů pro údaje s datem a časem:

Definice	Popis
DATE	Umožňuje uložení data od 1. ledna 1000 do 31. prosince 9999.
TIME	Umožňuje uložení času od –838:59:59 do 838:59:59.
TIMESTAMP	Slouží k uložení časového razítka od 1. ledna 1970 do roku 2037.
DATETIME	Umožňuje uložení data a času od 1. ledna 1000 do poslední sekundy 31. prosince 9999.
YEAR	Slouží k uložení čísla roku. Dvouciferným hodnotám roku se vyhýbejte, protože nejsou jednoznačné a automaticky se převádějí na čtyřciferný rok.

Dávejte pozor zejména při porovnávání hodnot typu `DATE` a `DATETIME`, protože každý pracuje jiným způsobem s časovou částí; výsledky porovnání mohou být neočekávané. Podrobné informace najdete opět v manuálu k MySQL, protože popsané chování se u různých verzí MySQL mírně liší.

Vytvoření tabulky

Databázový server nám běží, umíme přiřazovat uživatelská oprávnění a víme, jak vytvářet databáze a některé základní datové typy; můžeme se tedy podívat na další téma, a sice vytváření tabulek.

Tabulka v databázi není nic jiného než posloupnost řádků, z nichž každý se skládá z pevně definované množiny sloupců. Trochu se podobá tabulkovému sešitu, ale každý řádek zde musí mít stejný počet sloupců stejného typu, a navíc se každý řádek musí alespoň nějak odlišovat od všech ostatních řádků tabulky.

Databáze může obsahovat (v rozumných mezích) prakticky neomezený počet tabulek. Jen málokdy budeme ale v jedné databázi potřebovat více než 100 tabulek a u většiny malých databází si obvykle vystačíme s méně než čtvrtinou.

Úplná syntaxe příkazů SQL pro vytváření databázových objektů, kterým se souhrnně říká DDL (Data Definition Language, jazyk pro definici dat), je příliš složitá, než aby se nám vešla do textu této kapitoly, a proto vás pro podrobné informace opět odkážeme do dokumentace z webových stránek MySQL.

Základní syntaxe pro vytvoření tabulky vypadá takto:

```
CREATE TABLE <název_tabulky> (
sloupec typ [NULL | NOT NULL] [AUTO_INCREMENT] [PRIMARY KEY]
[, ...]
[, PRIMARY KEY (sloupec [, ...])]
)
```

Nepotřebnou tabulku můžeme odstranit velice jednoduchým příkazem `DROP TABLE`:

```
DROP TABLE <název_tabulky>
```

Do příkazu pro vytváření tabulek můžeme zadat celou řadu dalších klíčových slov, z nichž si nyní ukážeme ty nejdůležitější, které nám při práci hodně pomohou.

Klíčové slovo	Popis
AUTO_INCREMENT	Toto speciální klíčové slovo říká, že kdykoli do tohoto sloupce zapíšeme hodnotu NULL, doplní MySQL automaticky jeho hodnotu podle automaticky inkrementovaného čísla. Jedná se o velice užitečnou funkci, protože MySQL za nás takto automaticky přiřadí jedinečné číslo každému řádku tabulky; výraz je ale možné použít jen u sloupců, které jsou zároveň primárním klíčem. V jiných databázích zajišťuje podobnou funkci sériový číselný typ, automatické číslo nebo explicitně definovaná posloupnost.
NULL	Zvláštní databázová hodnota, která normálně vyjadřuje „neznámou" hodnotu, často ale znamená také „hodnota nemá smysl". Dejme tomu, že například vyplňujeme tabulku s informacemi o zaměstnancích, kde je sloupec pro osobní e-mailovou adresu, ale někteří zaměstnanci zkrátka osobní e-mail nemají. V takovém případě zapíšeme do sloupce hodnotu NULL a vyjádříme tak, že hodnota u daného zaměstnance nemá smysl. Zápis NOT NULL znamená, že do sloupce nelze zapsat hodnotu NULL a můžeme s ním zabránit v ukládání hodnot NULL – jinými slovy, takováto hodnota musí být vždy známá, například příjmení zaměstnance.
PRIMARY KEY	Indikuje, že data tohoto sloupce budou jedinečná a že se v nich tedy každý řádek bude lišit od všech ostatních řádků tabulky. Každá tabulka může mít jen jeden primární klíč.

Vyzkoušejte — Vytvoření tabulky a vložení dat

Praktické vytvoření tabulky je naštěstí mnohem snazší, než se učit popsanou základní syntaxi, takže si zkusíme společně vytvořit tabulku s názvem children, v níž bude ke každému dítěti uloženo jedinečné číslo, křestní jméno a věk. Číslo dítěte bude zároveň primárním klíčem.

1. Potřebný příkaz SQL vypadá takto:

```
CREATE TABLE children (
       childno INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY,
       fname VARCHAR(30),
       age INTEGER
);
```

   Všimněte si, že na rozdíl od většiny běžných programovacích jazyků se název sloupce (childno) zapisuje před příslušný typ sloupce (INTEGER).

2. Primární klíč je možné definovat také jiným zápisem, odděleně od vlastní definice sloupce; podívejme se na tuto alternativní syntaxi v následující interaktivní relaci:

```
mysql> use rick
Database changed
mysql> CREATE table children (
    -> childno INTEGER AUTO_INCREMENT NOT NULL,
```

```
    -> fname varchar(30),
    -> age INTEGER,
    -> PRIMARY KEY(childno)
    ->);
Query OK, 0 rows affected (0.04 sec)
mysql>
```

Všimněte si zápisu příkazu SQL přes několik řádků: MySQL nám pomocí výzvy -> naznačí, že se nacházíme na pokračovacím řádku. Dále si všimněte ukončení příkazu středníkem, o kterém jsme se už zmínili a kterým databázovému serveru říkáme, že může zahájit zpracování našeho požadavku.

V případě chyby můžeme v MySQL procházet předchozí příkazy, upravit je a znovu je zadat ke zpracování prostým stiskem Enter.

3. V tabulce máme tedy tabulku a nyní do ní zkusíme vkládat data. K tomu slouží v jazyce SQL příkaz INSERT. Protože sloupec childno jsme definovali jako AUTO_INCREMENT, neuvádíme u něj žádné konkrétní hodnoty, ale jedinečné číslo v něm přiřadí sám MySQL.

```
mysql> INSERT INTO children(fname, age) VALUES("Jenny", 21);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO children(fname, age) VALUES("Andrew", 17);
Query OK, 1 row affected (0.00 sec)
```

Správnost vložení dat zkontrolujeme tak, že je z databáze zkusíme přečíst. Nad tabulkou tedy provedeme příkaz SELECT:

```
mysql> SELECT childno, fname, age FROM children;
+---------+--------+------+
| childno | fname  | age  |
+---------+--------+------+
| 1       | Jenny  | 21   |
| 2       | Andrew | 17   |
+---------+--------+------+
2 rows in set (0.00 sec)

mysql>
```

Namísto úplného seznamu požadovaných sloupců můžeme zadat také zkratku *, která znamená výpis všech sloupců příslušné tabulky. Pro interaktivní účely je tento zápis velice šikovný, ale v ostrém, produkčním systému uvádějte jednotlivé sloupce vždy explicitně.

Jak příklad funguje

V příkladu jsme zahájili interaktivní relaci k databázovému serveru a přepnuli jsme se do databáze rick. Poté jsme napsali příkaz SQL pro vytvoření tabulky, přičemž jsme jej na obrazovce rozdělili do většího počtu řádků. Po ukončení příkazu SQL středníkem ; vytvořil MySQL požadovanou tabulku. Poté jsme příkazem INSERT vložili data, přičemž ve sloupci childno jsme ponechali automaticky přiřazená čísla. Nakonec jsme příkazem SELECT vybrali a zobrazili data z tabulky.

Na detailní rozbor jazyka SQL, který se víceméně týká návrhu databází, už není v této kapitole místo. Podrobnější informace najdete například na webových stránkách `www.mysql.com`. Případně doporučujeme knihu *SQL Kompletní průvodce* (Computer Press, 2005).

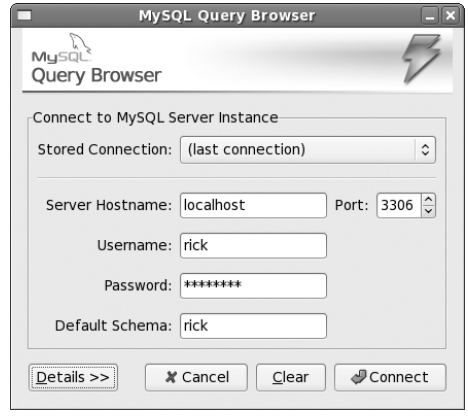
Grafické nástroje

Manipulace s tabulkami a daty na příkazovém řádku je sice v pořádku, ale dnes hodně lidí preferuje raději grafické nástroje.

MySQL má dva nejdůležitější grafické nástroje, a sice MySQL Administrator a MySQL Query Browser. Přesný název balíku k těmto nástrojům se liší podle konkrétní použité distribuce; v systému Red Hat hledejte například balíky `mysql-gui-tools` a `mysql-administrator` a v distribuci Ubuntu se nejprve podívejte do úložiště „Universe“ a potom vyhledejte balík `mysql-admin`.

MySQL Query Browser

Nástroj Query Browser neboli „prohlížeč dotazů“ je docela jednoduchý, ale mimořádně efektivní. Po instalaci jej můžeme vyvolat z grafické nabídky; po spuštění se objeví prvotní obrazovka s dotazem na spojovací informace, jak vidíme na obrázku 8.4.



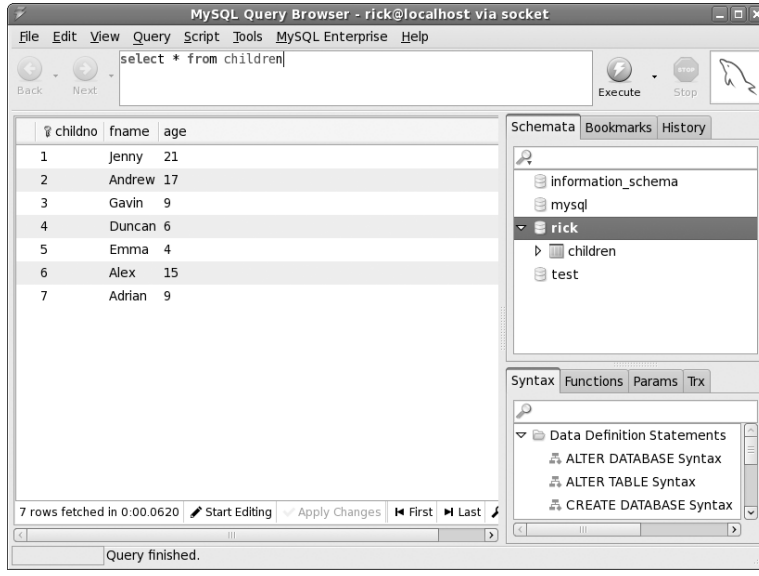
Obrázek 8.4

Pokud jste nástroj spustili na stejném počítači, kde běží i server, napište místo hostitelského názvu `localhost`.

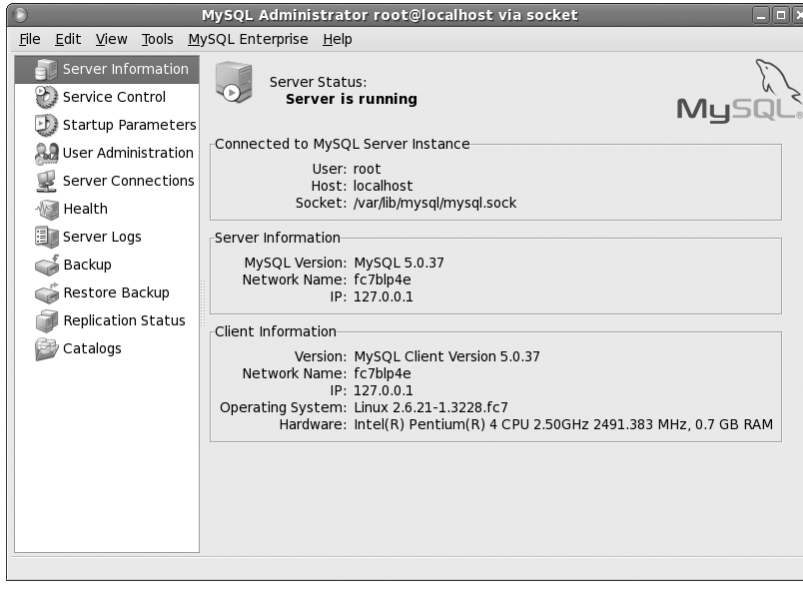
Po vlastním připojení se objeví jednoduché grafické rozhraní podle obrázku 8.5, z něhož můžeme spouštět dotazy v grafickém shellu; můžeme zde tedy využít možností grafické editace dotazu, dále grafické editace dat tabulky a také obrazovek s nápovědou a syntaxí příkazů SQL.

MySQL Administrator

Nástroj MySQL Administrator rozhodně nenechávejte ležet stranou, protože představuje velice silné, stabilní a snadno použitelné grafické rozhraní k systému MySQL, předem zkompilované pro platformy Linux a Windows (k dispozici je i zdrojový kód, pokud byste chtěli). Umožňuje administraci serveru MySQL a provádění příkazů SQL z grafického rozhraní.



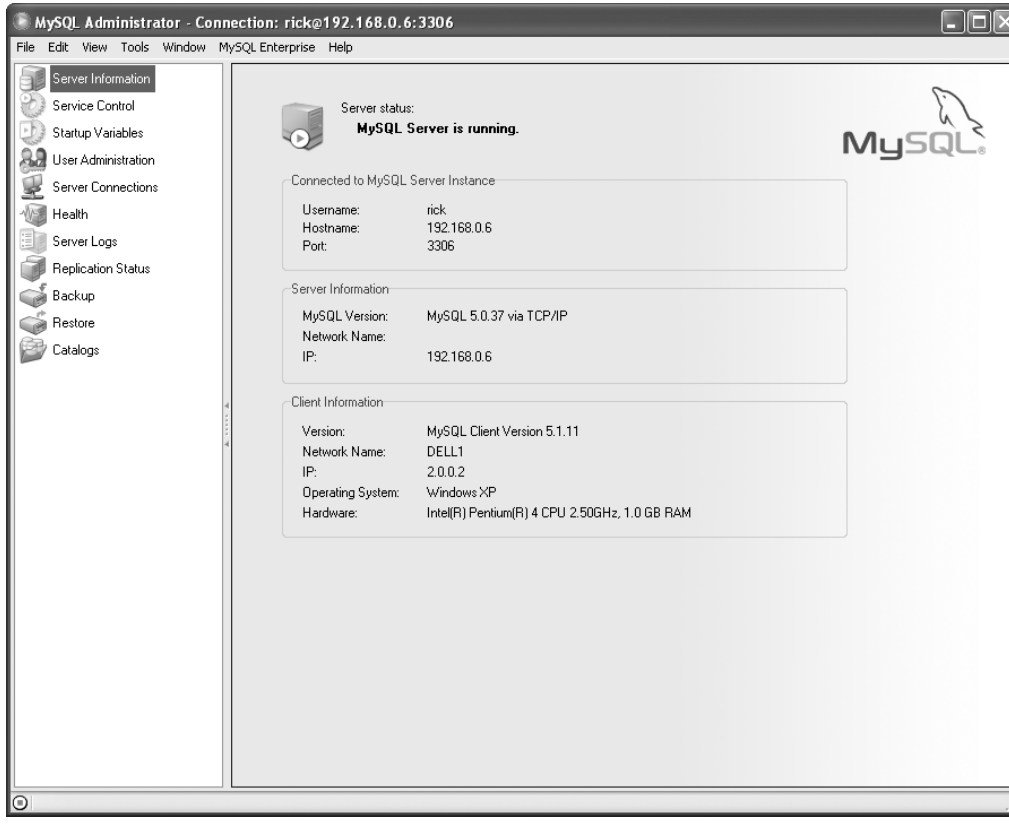
Obrázek 8.5



Obrázek 8.6

Po spuštění MySQL Administratoru se objeví připojovací obrazovka podobná úvodní obrazovce MySQL Query Browseru. Po zadání potřebných informací se dostaneme na hlavní ovládací stránku, která je na obrázku 8.6.

Chcete-li provádět administraci serveru MySQL z klienta Windows, můžete si z webových stránek MySQL, ze sekce grafických nástrojů stáhnout verzi MySQL Administratora pro Windows. V době vzniku této knihy zde byl k dispozici Administrator, Query Browser a utilita pro převod (migraci) databází. Stavová obrazovka je na obrázku 8.7; jak vidíte, je téměř totožná s linuxovou verzí.



Obrázek 8.7

> Pamatujte si, že pokud jste postupovali podle instrukcí pro zabezpečení serveru MySQL, může se `root` připojit jen z lokálního počítače `localhost`, nikoli z jiných počítačů v síti.

Jakmile nástroj MySQL Administrator běží, můžete se v něm podívat na různé možnosti konfigurace a monitorování systému. Práce s nástrojem je velice jednoduchá, ale my se mu v této kapitole již podrobněji věnovat nebudeme.

Přístup k datům MySQL z jazyka C

Základy MySQL jsme zvládli a nyní se můžeme pustit do přístupu k datům z aplikací, nikoli z grafických nástrojů či ze základního klienta `mysql`.

K databázím MySQL je možné přistupovat z mnoha různých programovacích jazyků, například:

- C,
- C++,
- Java,
- Perl,
- Python,
- Eiffel,
- Tcl,
- Ruby,
- PHP.

Navíc je k dispozici také ovladač ODBC, s nímž můžeme k databázi MySQL přistupovat z nativních aplikací Windows, jako je Access; existuje dokonce i ovladač ODBC pro Linux, i když pro jeho využití není příliš mnoho důvodů.

V této kapitole se omezíme jen na rozhraní jazyka C, protože na ten je zaměřena celá knížka a navíc protože stejné knihovny pak využíváme i pro připojení z řady jiných jazyků.

Rutiny pro připojení

Z jazyka C se k databázi MySQL připojíme v následujících dvou krocích:

- inicializace struktury popisovače připojení (connection handle),
- fyzické vytvoření připojení.

Nejprve provedeme inicializaci popisovače připojení, a to pomocí funkce `mysql_init`:

```
#include <mysql.h>

MYSQL *mysql_init(MYSQL *);
```

Normálně do parametru této funkce předáváme hodnotu `NULL` a ve výsledku se vrací ukazatel na nově alokovanou strukturu popisovače připojení. Předáme-li existující strukturu, provede se její opakovaná inicializace. Při chybě vrátí funkce hodnotu `NULL`.

Máme tedy alokovanou a inicializovanou strukturu. Ještě ale potřebujeme definovat další parametry připojení, které předáváme do funkce `mysql_real_connect`:

```
MYSQL *mysql_real_connect(MYSQL *connection,
       const char *server_host,
       const char *sql_user_name,
       const char *sql_password,
       const char *db_name,
       unsigned int port_number,
       const char *unix_socket_name,
       unsigned int flags);
```

Ukazatel na připojení musí definovat strukturu, kterou jsme dříve inicializovali ve volání `mysql_init`. Ostatní parametry nepotřebují bližší vysvětlení; pouze poznamenejme, že parametrem `server_host`

může být hostitelský název nebo IP adresa počítače. Při připojení k lokálnímu počítači stačí zadat text `localhost` a tak optimalizovat typ připojení.

Význam parametrů `sql_user_name` a `sql_password` je zřejmý. Pokud je přihlašovací jméno prázdné (`NULL`), předpokládá se přihlašovací ID aktuálního uživatele systému Linux. Pokud je heslo prázdné (`NULL`), můžeme na serveru přistupovat jen k těm datům, která jsou dostupná i bez hesla. Heslo se před odesláním po síti šifruje.

Do dvojice parametrů `port_number` a `unix_socket_name` zapisujeme hodnoty `0` a `NULL`, nezměníme-li v instalaci MySQL příslušné výchozí hodnoty. Výchozí hodnotou parametrů je vždy odpovídající hodnota.

Poslední parametr `flags` obsahuje bitový vzorek příznaků `#define`, kombinovaných pomocí logického součtu `OR`, a slouží ke změně jistých vlastností komunikačního protokolu. Pro účely této úvodní kapitoly nemají příznaky význam, a pokud vás zajímá jejich úplná dokumentace, podívejte se do manuálu.

Pokud se funkce nemůže připojit, vrátí hodnotu `NULL`. Bližší informace pak dostaneme z funkce `mysql_error`.

Jakmile práci se spojením skončíte nebo jakmile se dostanete na konec programu, vyvolejte funkci `mysql_close`:

```
void mysql_close(MYSQL *connection);
```

Toto volání ukončí činnost zadaného připojení. Pokud bylo připojení vytvořeno ve volání `mysql_init`, struktura se uvolní; ukazatel se stává neplatným a nemůžeme s ním znovu pracovat. Ponechat v otevřeném stavu nepotřebné připojení je plýtvání prostředky, ale na druhé straně i jeho opětovné otevření znamená jistou režii, takže obě funkce používejte s rozmyslem.

Rutina `mysql_options`, kterou je možné vyvolat jen mezi voláními `mysql_init` a `mysql_real_connect`, slouží k nastavení jistých voleb:

```
int mysql_options(MYSQL *connection, enum option_to_set,
                  const char *argument);
```

Protože v jednom volání umí tato funkce nastavit vždy pouze jedinou volbu, musíme ji při nastavení několika voleb spouštět opakovaně. Její opakované spuštění není problém – jen se musíme vejít mezi zmíněné funkce `mysql_init` a `mysql_real_connect`. Ne všechny volby jsou ve skutečnosti typu `char`, takže je musíme přetypovat na `const char *`. Tři nejběžnější volby uvádí následující tabulka; všechny jsou jako vždy uvedeny v obsáhlém online manuálu.

Volba – hodnota enum	Skutečný typ argumentu	Popis
`MYSQL_OPT_CONNECT_TIMEOUT`	`const unsigned int *`	Počet sekund, po které se má nejdéle čekat a po jejichž uplynutí končí časový limit připojení.
`MYSQL_OPT_COMPRESS`	Není, zadejte `NULL`	Nad síťovým připojením bude probíhat komprese.
`MYSQL_INIT_COMMAND`	`const char *`	Příkaz, který se bude odesílat při každém navázání připojení.

Z úspěšného volání se vrací nula. Protože funkce slouží jen pro nastavení jistých příznaků, selhání vždy znamená, že jsme zadali neplatnou volbu.

V následujícím fragmentu kódu nastavíme časový limit připojení na 7 sekund:

```
unsigned int timeout = 7;
...
connection = mysql_init(NULL);
ret = mysql_options(connection, MYSQL_OPT_CONNECT_TIMEOUT,
                    (const char *)&timeout);
if (ret) {
    /* Ošetření chyby */
    ...
}
connection = mysql_real_connect(connection ...
```

Už tedy umíme vytvořit a uzavřít připojení a můžeme si všechno vyzkoušet v krátkém programu.

Začneme definicí nového hesla pro uživatele `rick` na počítači `localhost` (v následujícím kódu) a poté vytvořením databáze `foo`, k níž se budeme připojovat. Všechny operace už dobře znáte, takže je jenom sepíšeme dohromady:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> GRANT ALL ON *.* TO rick@localhost IDENTIFIED BY 'tajné';
Query OK, 0 rows affected (0.01 sec)

mysql> \q
Bye
$ mysql -u rick -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> CREATE DATABASE foo;
Query OK, 1 row affected (0.01 sec)

mysql> \q
```

Tím jsme vytvořili novou databázi. Abychom nemuseli všechny příkazy pro vytvoření tabulky a pro její naplnění daty zapisovat přímo na příkazový řádek `mysql`, což je hodně pracné (zejména v případě opakování) a hrozí velké riziko chyb, vytvoříme soubor se všemi potřebnými příkazy.

Soubor se nazývá `create_children.sql` a vypadá takto:

```
--
-- Vytvoření tabulky children
--
```

```
CREATE TABLE children (
    childno int(11) NOT NULL auto_increment,
    fname varchar(30),
    age int(11),
    PRIMARY KEY (childno)
);

--
-- Naplnění tabulky 'children'
--

INSERT INTO children(childno, fname, age) VALUES (1,'Jenny',21);
INSERT INTO children(childno, fname, age) VALUES (2,'Andrew',17);
INSERT INTO children(childno, fname, age) VALUES (3,'Gavin',8);
INSERT INTO children(childno, fname, age) VALUES (4,'Duncan',6);
INSERT INTO children(childno, fname, age) VALUES (5,'Emma',4);
INSERT INTO children(childno, fname, age) VALUES (6,'Alex',15);
INSERT INTO children(childno, fname, age) VALUES (7,'Adrian',9);
```

Nyní se můžeme znovu přihlásit k MySQL, vybrat databázi `foo` a spustit tento soubor. Pro stručnost (a také jako příklad, který můžete přímo zapsat do skriptu) uvádíme heslo přímo na příkazovém řádku:

```
$ mysql -u rick --password=tajné foo
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> \. create_children.sql
Query OK, 0 rows affected (0.01 sec)

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

Z výsledků jsme odstranili opakované řádky, které se vypisují při vložení každého řádku do databáze. To znamená, že máme nejen uživatele, databázi a tabulku, ale už v ní také máme nějaká data – a můžeme k nim tedy zkusit přistupovat z kódu.

Ukážeme si program `connect1.c`, který se pod uživatelským jménem `rick` a heslem `tajné` připojí k serveru na lokálním počítači a v něm k databázi `foo`:

```
#include <stdlib.h>
#include <stdio.h>

#include "mysql.h"

int main(int argc, char *argv[]) {
    MYSQL *conn_ptr;

    conn_ptr = mysql_init(NULL);
    if (!conn_ptr) {
        fprintf(stderr, "mysql_init selhalo\n");
        return EXIT_FAILURE;
```

```
    }

    conn_ptr = mysql_real_connect(conn_ptr, "localhost", "rick", "tajné",
                                  "foo", 0, NULL, 0);
    if (conn_ptr) {
    printf("Připojení je úspěšné\n");
    } else {
    printf("Připojení selhalo\n");
    }

    mysql_close(conn_ptr);

    return EXIT_SUCCESS;
}
```

Program zkompilujeme a uvidíme, co se bude dít. Při kompilaci budeme možná muset uvést cestu k souborům `include` i ke knihovnám, a také nařídit sestavení programu s knihovním modulem `mysqlclient`. V některých systémech musíme doplnit také volbu `-lz`, která linkuje také kompresní knihovnu. Na systému autora bylo nutné ke kompilaci napsat tento příkazový řádek:

```
$ gcc -I/usr/include/mysql connect1.c -L/usr/lib/mysql -lmysqlclient -o connect1
```

Pro úspěšnou kompilaci bude zřejmě také nutné zkontrolovat, jestli máme nainstalované příslušné klientské balíky a v jakých jsou umístěních, a odpovídajícím způsobem upravit předchozí příkazový řádek.

Jakmile program spustíme, měla by se objevit zpráva o úspěšném připojení:

```
$ ./connect1
Připojení je úspěšné
$
```

V kapitole 9 si ukážeme, jak tuto kompilaci automatizovat pomocí vhodného souboru `makefile`.

Jak vidíte, připojit se k databázi MySQL je velice snadné.

Ošetření chyb

Než se pustíme do složitějších programů, podíváme se ještě, jakým způsobem MySQL ošetřuje chyby. Struktura popisovače připojení (connection handle) oznamuje v MySQL celou řadu návratových chybových kódů. Dvě nejdůležitější rutiny jsou:

```
unsigned int mysql_errno(MYSQL *connection);
```

a

```
char *mysql_error(MYSQL *connection);
```

Chybový kód má obvykle nenulovou hodnotu a zjistíme jej z volání `mysql_errno`, do něhož předáme strukturu připojení. Nula se vrací v případě, že chybový kód nebyl definován. Protože kód se aktualizuje při každém volání funkce z knihovny, dostaneme vždy chybový kód pouze k po-

slední provedené funkci, samozřejmě jen s výjimkou dvou uvedených chybových funkcí, které chybový kód neaktualizují.

Návratovou hodnotu tvoří samotný chybový kód a jeho hodnoty jsou definovány v include souboru `errmsg.h` nebo `mysqld_error.h`; oba soubory se nacházejí v adresáři `include` instalace MySQL. První soubor definuje chyby na straně klienta, zatímco druhý vysvětluje serverové chyby.

Máte-li raději textovou podobu chybových zpráv, můžete zavolat funkci `mysql_error`, která vrací namísto kódu čitelný text. Při volání se ale text zapíše do interního statického paměťového prostoru, takže jej pro pozdější využití musíte zkopírovat jinam.

Abychom si všechno ukázali v praxi, musíme do našeho programového kódu doplnit jisté základní ošetření chyb. Nejspíše jste už ale postřehli, že u funkce `mysql_real_connect` máme jistý problém – ta při selhání vrací nulový ukazatel a chybový kód nám „nedopřeje“. Jestliže z popisovače připojení vytvoříme proměnnou, dostaneme se k němu i při selhání funkce `mysql_real_connect`.

Dostáváme se tedy k verzi `connect2.c`, v níž si ilustrujeme práci se strukturou připojení, která není dynamicky alokovaná, a také základní ošetření chyb. Všechny změny jsou zvýrazněny:

```
#include <stdlib.h>
#include <stdio.h>

#include "mysql.h"

int main(int argc, char *argv[]) {
    MYSQL my_connection;

    mysql_init(&my_connection);
    if (mysql_real_connect(&my_connection, "localhost", "rick",
                           "Nevím", "foo", 0, NULL, 0)) {
        printf("Připojení je úspěšné\n");
        mysql_close(&my_connection);
   } else {
        fprintf(stderr, "Připojení selhalo\n");
        if (mysql_errno(&my_connection)) {
            fprintf(stderr, "Chyba připojení %d: %s\n",
            mysql_errno(&my_connection), mysql_error(&my_connection));
        }
   }

    return EXIT_SUCCESS;
}
```

Problém bychom mohli vyřešit ještě snáze, a sice tak, že bychom při selhání `mysql_real_connect` zkrátka nepřepisovali ukazatel připojení návratovou hodnotou. Přesto je příklad pěknou ukázkou práce se strukturami připojení. Chybový stav zde můžeme vynutit zadáním nesprávného uživatelského jména či hesla a opět dostaneme podobný chybový kód, jaký nabízí nástroj `mysql`.

```
$ ./connect2
Připojení selhalo
```

```
Connection error 1045: Access denied for user: 'rick@localhost' (Using
password: YES)
$
```

Provádění příkazů SQL

Umíme se tedy připojit k databázi a korektně ošetřit chyby, takže je na čase zkusit v programu provádět také nějakou užitečnou práci. Nejdůležitější funkce rozhraní API provádí příkazy SQL, jak ostatně i její název napovídá:

```
int mysql_query(MYSQL *connection, const char *query)
```

Není to nijak těžké, že? Rutina přebírá ukazatel na strukturu připojení a textový řetězec s příkazem SQL – doufejme že platným (na rozdíl od nástroje `mysql` zde ale chybí středník na konci). Po úspěšném dokončení vrací nulu. Druhá rutina, `mysql_real_query`, slouží pro načítání binárních dat; my ale v této kapitole budeme pracovat jen s funkcí `mysql_query`.

Příkazy SQL, které nevracejí žádná data

Pro jednoduchost začneme několika příkazy SQL, které nevracejí žádná data, a sice `UPDATE`, `DELETE` a `INSERT`.

Na tomto místě si musíme představit další důležitou funkci, která zjistí, kolik řádků bylo dotčeno zpracováním posledního dotazu:

```
my_ulonglong mysql_affected_rows(MYSQL *connection);
```

Jako první si asi na této funkci všimneme velmi neobvyklého datového typu návratové hodnoty; z důvodů lepší přenositelnosti je to typ bez znaménka a při výpisu pomocí funkce `printf` se doporučuje jej přetypovat na `unsigned long` s formátem `%lu`. Zmíněná funkce vrací počet řádků, dotčených předchozím dotazem `UPDATE`, `INSERT` nebo `DELETE`. Pokud jste zvyklí pracovat s jinými databázemi SQL, může vás návratová hodnota hodně překvapit – MySQL vrací totiž počet řádků, které byly při aktualizaci skutečně modifikovány, zatímco jiné databáze považují za aktualizovaný každý záznam, jenž odpovídá klauzuli `WHERE`.

Stejně jako u všech funkcí `mysql_` návratová hodnota 0 znamená, že dotazem nebyly dotčeny žádné řádky, zatímco kladné číslo je skutečný výsledek, zpravidla počet řádků.

Nejprve musíme v databázi `foo` vytvořit tabulku `children` (pokud zde ještě není). Příkazem `drop` odstraníme případnou stávající tabulku, abychom tak ve výsledku měli čistou definici a aby se obnovila posloupnost identifikátorů ve sloupci typu `AUTO_INCREMENT`:

```
$ mysql -u rick -p foo
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> DROP TABLE children;
Query OK, 0 rows affected (0.58 sec)

mysql> CREATE TABLE children (
    -> childno int(11) AUTO_INCREMENT NOT NULL PRIMARY KEY,
```

```
    -> fname varchar(30),
    -> age int
    ->);
Query OK, 0 rows affected (0.09 sec)
mysql>
```

Poté do programu `connect2.c` doplníme další kód, který do naší tabulky bude vkládat nový řádek; program nazveme `insert1.c`. Všimněte si, že zalomení řádků je vynuceno pouze rozložením tištěných stránek knížky; normálně v příkazech SQL řádky nezalamujeme, jen s výjimkou velmi dlouhých příkazů, kde na konci zlomeného řádku napíšeme `\`.

```
#include <stdlib.h>
#include <stdio.h>

#include "mysql.h"

int main(int argc, char *argv[]) {
    MYSQL my_connection;
    int res;

    mysql_init(&my_connection);
    if (mysql_real_connect(&my_connection, "localhost",
                           "rick", "tajné", "foo", 0, NULL, 0)) {
        printf("Připojení je úspěšné\n");

        res = mysql_query(&my_connection, "INSERT INTO children(fname, age)
                          VALUES('Ann', 3)");
        if (!res) {
            printf("Vloženo %lu řádků\n",
                  (unsigned long)mysql_affected_rows(&my_connection));
       } else {
            fprintf(stderr, "Chyba při vložení %d: %s\n",
                    mysql_errno(&my_connection),
                    mysql_error(&my_connection));
       }

        mysql_close(&my_connection);
       } else {
            fprintf(stderr, "Připojení selhalo\n");
            if (mysql_errno(&my_connection)) {
            fprintf(stderr, "Chyba připojení %d: %s\n",
                    mysql_errno(&my_connection), mysql_error(&my_connection));
       }
   }
    return EXIT_SUCCESS;
}
```

Není velkým překvapením, že se do tabulky skutečně vloží jeden řádek.

Nyní kód ještě jednou změníme, namísto operace INSERT budeme provádět UPDATE a opět se podíváme na zprávu o počtu dotčených řádků.

```
  mysql_errno(&my_connection), mysql_error(&my_connection));
      }
   }

    res = mysql_query(&my_connection, "UPDATE children SET AGE = 4
                                       WHERE fname = 'Ann'");
    if (!res) {
        printf("Aktualizováno %lu řádků\n",
              (unsigned long)mysql_affected_rows(&my_connection));
   } else {
        fprintf(stderr, "Chyba při aktualizaci %d: %s\n",
              mysql_errno(&my_connection),
              mysql_error(&my_connection));
   }
```

Tuto verzi programu nazveme update1.c; ve svém kódu se pokouší přiřadit všem dětem se jménem Ann věk 4 roky.

Uvažujme, že jsou v tabulce dětí zapsány následující údaje:

```
mysql> SELECT * from CHILDREN;
+---------+--------+------+
| childno | fname  | age  |
+---------+--------+------+
| 1       | Jenny  | 21   |
| 2       | Andrew | 17   |
| 3       | Gavin  | 9    |
| 4       | Duncan | 6    |
| 5       | Emma   | 4    |
| 6       | Alex   | 15   |
| 7       | Adrian | 9    |
| 8       | Ann    | 3    |
| 9       | Ann    | 4    |
| 10      | Ann    | 3    |
| 11      | Ann    | 4    |
+---------+--------+------+
11 rows in set (0.00 sec)
```

Všimněte si, že jméno Ann tu mají hned čtyři děti. Po spuštění programu update1 vás proto možná napadne, že by i počet dotčených řádků měl být roven čtyřem, protože tolik jich splňuje podmínku klauzule WHERE. Ve skutečnosti ale uvidíme, že program změní jen dva řádky – pouze v nich bylo totiž nutné data změnit. Chcete-li se vrátit k „tradičnímu“ počítání řádků, nastavte ve volání mysql_real_connect nastavit příznak CLIENT_FOUND_ROWS:

```
if (mysql_real_connect(&my_connection, "localhost",
                "rick", "tajné", "foo", 0, NULL, CLIENT_FOUND_ROWS)) {
```

Vrátíte-li nyní data v databázi zpět do původní podoby a spustíte-li program s upraveným příznakem, oznámí již čtyři dotčené řádky.

Funkce `mysql_affected_rows` má ještě jednu zvláštnost, která se projeví při odstranění dat z databáze. Odstraníte-li data pomocí klauzule `WHERE`, vrátí zmíněná funkce v souladu s očekáváním počet odstraněných řádků. Pokud ale příkaz `DELETE` žádnou klauzuli `WHERE` neobsahuje, odstraní se správně všechny řádky tabulky, ovšem funkce oznámí do programu počet dotčených řádků nula. To je proto, že operaci odstranění všech řádků stroj MySQL optimalizuje a neprovádí tedy řadu jednotlivých odstranění. Na uvedené chování nemá vliv ani nastavení příznaku `CLIENT_FOUND_ROWS`.

Jak zjistit, co jsme vložili

S vkládáním dat je spojena jedna drobná, ale podstatná věc. Vzpomínáte si na sloupec typu `AUTO_INCREMENT`, do něhož MySQL vkládá číselné identifikátory automaticky? Tato funkce je velice užitečná, zejména při současné práci více uživatelů.

Podívejme se ale na definici tabulky ještě jednou:

```
CREATE TABLE children (
       childno INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY,
       fname VARCHAR(30),
       age INTEGER
);
```

Jak vidíme, sloupec `childno` je polem typu `AUTO_INCREMENT`. To je v pořádku, ale když do tabulky vložíme řádek, jak zjistíme, pod jakým číslem bylo dítě do tabulky zapsáno?

Nad tabulkou bychom mohli provést příkaz `SELECT` a vyhledat dítě podle jména, takový dotaz je ale velmi neefektivní a navíc nemusí vrátit jednoznačnou odpověď – i v posledním příkladu bylo několik dětí stejného jména. A co když několik uživatelů pořizuje rychle nová data, takže od vložení našeho řádku po příkaz `SELECT` proběhlo vložení ještě několika dalších řádků dat? Problém zjištění hodnoty sloupce `AUTO_INCREMENT` je velice častý, a proto MySQL nabízí speciální řešení, kterým je funkce `LAST_INSERT_ID()`.

Kdykoli MySQL vloží nějakou datovou hodnotu do sloupce typu `AUTO_INCREMENT`, sleduje pro každého uživatele zvlášť poslední takto přiřazenou hodnotu. Uživatelský program zjistí tuto hodnotu prostým dotazem `SELECT` nad speciální funkcí `LAST_INSERT_ID()`, která takto funguje jako jakýsi pseudo-sloupec.

Vyzkoušejte — Zjištění identifikátoru vygenerovaného z AUTO_INCREMENT

Všechno si ukážeme v praxi – do tabulky vložíme nějaké hodnoty a poté zavoláme funkci `LAST_INSERT_ID()`.

```
mysql> INSERT INTO children(fname, age) VALUES('Tom', 13);
Query OK, 1 row affected (0.06 sec)
mysql> SELECT LAST_INSERT_ID();
+------------------+
```

```
| last_insert_id() |
+------------------+
| 14               |
+------------------+
1 row in set (0.01 sec)
mysql> INSERT INTO children(fname, age) VALUES('Harry', 17);
Query OK, 1 row affected (0.02 sec)
mysql> SELECT LAST_INSERT_ID();
+------------------+
| last_insert_id() |
+------------------+
| 15               |
+------------------+
1 row in set (0.00 sec)
mysql>
```

Jak příklad funguje

S každým vložením nového řádku přiřadil MySQL novou hodnotu sloupce `id` a zapamatoval si ji, takže jsme ji snadno zjistili voláním funkce `LAST_INSERT_ID()`.

Chcete-li nyní trochu experimentovat a ověřit si, že je navrácené číslo skutečně jedinečné pro danou uživatelskou relaci, otevřete si jinou relaci a vložte další řádek. V původní relaci znovu zadejte příkaz `SELECT LAST_INSERT_ID();` a uvidíte, že se výsledek nezměnil – stále jím je číslo, naposledy vložené v této relaci. Jakmile ale provedete příkaz `SELECT * FROM children`, uvidíte již nově vložené řádky dat.

Vyzkoušejte Automatické identifikátory v jazyce C

V tomto příkladu upravíme program `insert1.c` a ukážeme si, jak automatické identifikátory fungují z programu v C. Podstatné změny jsou opět zvýrazněny; program nazveme `insert2.c`.

```
#include <stdlib.h>
#include <stdio.h>

#include "mysql.h"

int main(int argc, char *argv[]) {
    MYSQL my_connection;
    MYSQL_RES *res_ptr;
    MYSQL_ROW sqlrow;
    int res;

    mysql_init(&my_connection);
    if (mysql_real_connect(&my_connection, "localhost",
                           "rick", "tajné", "foo", 0, NULL, 0)) {
        printf("Připojení je úspěšné\n");
```

```
        res = mysql_query(&my_connection, "INSERT INTO children(fname, age)
                                           VALUES('Robert', 7)");

        if (!res) {
            printf("Vloženo %lu řádků\n", (unsigned
            long)mysql_affected_rows(&my_connection));
        } else {
        fprintf(stderr, "Chyba při vložení %d: %s\n", mysql_errno(&my_connection),
                         mysql_error(&my_connection));
        }

        res = mysql_query(&my_connection, "SELECT LAST_INSERT_ID()");

        if (res) {
            printf("Chyba při SELECT: %s\n", mysql_error(&my_connection));
        } else {
            res_ptr = mysql_use_result(&my_connection);
            if (res_ptr) {
                while ((sqlrow = mysql_fetch_row(res_ptr))) {
                    printf("Vložili jsme childno %s\n", sqlrow[0]);
                }
                mysql_free_result(res_ptr);
            }
        }

        mysql_close(&my_connection);
    } else {
        fprintf(stderr, "Připojení selhalo\n");
        if (mysql_errno(&my_connection)) {
        fprintf(stderr, "Chyba připojení %d: %s\n",
                mysql_errno(&my_connection), mysql_error(&my_connection));
        }
    }

    return EXIT_SUCCESS;
}
```

A tady jsou výsledky:

```
$ gcc -I/usr/include/mysql insert2.c -L/usr/lib/mysql -lmysqlclient -o insert2
$ ./insert2
Připojení je úspěšné
Vloženo 1 řádků
Vložili jsme childno 6
$ ./insert2
Připojení je úspěšné
```

```
Vloženo 1 řádků
Vložili jsme childno 7
```

Jak příklad funguje

Po vložení řádku jsme pomocí funkce `LAST_INSERT_ID()`, zapsané do normálního příkazu `SELECT`, zjistili nově přiřazený identifikátor. Poté jsme ve funkci `mysql_use_result()`, o níž si povíme za chvíli, načetli data z provedeného příkazu `SELECT` a vypsali je. Mechanismem načtení hodnoty se příliš nezabývejte; všechno si vysvětlíme na následujících stránkách.

Příkazy, které data vracejí

Mnohem častěji ale pomocí jazyka SQL data načítáme, než je vkládáme a aktualizujeme. A k operaci načítání v něm slouží příkaz `SELECT`.

> MySQL podporuje pro navracení výsledků také příkazy `SHOW`, `DESCRIBE` a `EXPLAIN`, kterým se ale v tomto výkladu věnovat nebudeme. Pokud vás zajímají blíže, podívejte se do příslušné dokumentace.

Načítání dat do aplikace v jazyce C se obvykle bude skládat ze čtyř kroků:

- zadání dotazu,
- načtení dat,
- zpracování dat,
- závěrečný úklid (podle potřeby).

Zadání dotazu SQL probíhá, stejně jako u příkazů `INSERT` a `DELETE`, ve funkci `mysql_query`. Dále musíme data načíst, a to podle požadovaného způsobu načtení vyvoláním funkce `mysql_store_result` nebo `mysql_use_result`. V dalším kroku přistoupíme ke zpracování dat pomocí série volání `mysql_fetch_row`. A nakonec prostřednictvím funkce `mysql_free_result` uvolníme paměť, kterou jsme v dotazu používali.

Rozdíl mezi funkcemi `mysql_use_result` a `mysql_store_result` je v podstatě ekvivalentní tomu, jestli budeme chtít data vracet po jednotlivých řádcích, nebo chceme celou výslednou množinu najednou. Druhý uvedený postup je vhodnější zejména v případě, že očekáváme menší výslednou množinu.

Funkce pro hromadné načtení dat

Veškerá data z výsledků příkazu `SELECT` (nebo jiného příkazu, jenž vrací data) můžeme načíst v jediném volání funkce `mysql_store_result`:

```
MYSQL_RES *mysql_store_result(MYSQL *connection);
```

Je zřejmé, že tuto funkci má smysl volat jen po úspěšném dokončení funkce `mysql_query`. Funkce okamžitě uloží veškerá data, která se klientovi vrátila; jako výsledek vrací ukazatel na novou strukturu výsledné množiny (result set), respektive při selhání příkazu `NULL`.

Při úspěchu lze v následném volání `mysql_num_rows` zjistit počet navrácených záznamů, který by v zásadě měl být kladný; pokud dotaz nevrátí žádné výsledky, může být roven nule.

```
my_ulonglong mysql_num_rows(MYSQL_RES *result);
```

Tato funkce přebírá výslednou strukturu, navrácenou z volání funkce `mysql_store_result`, a vrací počet řádků v této výsledné množině. Pokud funkce `mysql_store_result` skončila úspěchem, bude i volání `mysql_num_rows` úspěšné.

Pomocí této kombinace funkcí můžeme snadno načíst potřebná data. V tomto okamžiku jsou data vůči klientu lokální a nemusíme se tedy zabývat možností chyb v síti či v databázi. Načtením počtu navrácených řádků se připravíme na následné programování.

Pracujete-li s hodně velkou datovou množinou, bude vhodnější načítat výsledky po menších, lépe zvládnutelných blocích informací. Tím se může rychleji vrátit řízení do aplikace a zároveň nemusíme zbytečně obsazovat síťové prostředky. K této myšlence se ještě vrátíme hlouběji, v souvislosti s funkcí `mysql_use_result`.

Data máme tedy načtena a můžeme je zpracovat pomocí funkce `mysql_fetch_row`, přičemž v datové množině se pohybujeme s funkcemi `mysql_data_seek`, `mysql_row_seek` a `mysql_row_tell`. Podívejme se, co která funkce dělá:

- Funkce `mysql_fetch_row` načte jeden řádek dat z výsledné struktury, navrácené ve funkci `mysql_store_result`, a zapíše jej do struktury řádku. Při vyčerpání dat nebo při vzniku chyby vrací `NULL`. Ke zpracování dat ve struktuře řádku se vrátíme v následující části textu.

  ```
  MYSQL_ROW mysql_fetch_row(MYSQL_RES *result);
  ```

- Tato funkce, `mysql_data_seek`, umožňuje přechod na jiné místo výsledné množiny a nastavuje ji tak na řádek, který se vrátí v nejbližší operaci `mysql_fetch_row`. Hodnota offsetu je rovna číslu řádku a musí se pohybovat od 0 do čísla o jedničku menšího, než je počet řádků výsledné množiny. Předáme-li do parametru nulu, vrátí se při příštím volání `mysql_fetch_row` první řádek dat.

  ```
  void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset);
  ```

- Pomocí funkce `mysql_row_tell` zjistíme hodnotu offset, která indikuje aktuální pozici ve výsledné množině. Nejedná se ale o číslo řádku a nemůžeme ji tedy předávat ani do funkce `mysql_data_seek`.

  ```
  MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result);
  ```

- Můžeme ji ale využít ve volání

  ```
  MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset);
  ```

  které přesune aktuální pozici ve výsledné množině a zároveň vrátí předchozí pozici.

  Uvedená dvojice funkcí je nejužitečnější pro přesun mezi známými pozicemi výsledné množiny. Dávejte ale pozor a nepleťte si hodnotu offsetu z funkcí `row_tell` a `row_seek` s číslem `row_number`, které se používá ve volání `data_seek`. Po záměně hodnot budou výsledky nepředvídatelné.

- Jakmile veškeré potřebné operace s daty skončíte, nezapomeňte na explicitní volání `mysql_free_result`, ve kterém knihovna MySQL po sobě „uklidí“.

  ```
  void mysql_free_result(MYSQL_RES *result);
  ```

  Po dokončení prací s výslednou množinou je potřeba tuto funkci vždy vyvolat, aby knihovna MySQL mohla správně ukončit všechny alokované objekty.

Načtení dat

Konečně se dostáváme k napsání naší první aplikace pro načítání dat. Vybereme záznamy všech dětí, jejichž věk je větší než 5. Data zatím neumíme zpracovat, a proto je pouze načteme. Zvýrazněna je důležitá část programu, v němž načteme výslednou množinu a v cyklu ji procházíme. Program nazveme `select1.c`:

```
#include <stdlib.h>
#include <stdio.h>

#include "mysql.h"

MYSQL my_connection;
MYSQL_RES *res_ptr;
MYSQL_ROW sqlrow;

int main(int argc, char *argv[]) {
    int res;

    mysql_init(&my_connection);
    if (mysql_real_connect(&my_connection, "localhost", "rick",
                           "tajné", "foo", 0, NULL, 0)) {
        printf("Připojení je úspěšné\n");

        res = mysql_query(&my_connection, "SELECT childno, fname,
                          age FROM children WHERE age > 5");

        if (res) {
            printf("Chyba při SELECT: %s\n", mysql_error(&my_connection));
       } else {
            res_ptr = mysql_store_result(&my_connection);
            if (res_ptr) {
                printf("Načteno %lu řádků\n",
                        (unsigned long)mysql_num_rows(res_ptr));
                while ((sqlrow = mysql_fetch_row(res_ptr))) {
                    printf("Načtená data...\n");
               }
                if (mysql_errno(&my_connection)) {
                    fprintf(stderr, "Chyba při načítání: %s\n",
                           mysql_error(&my_connection));
               }
                mysql_free_result(res_ptr);}
          }
        mysql_close(&my_connection);

       } else {
            fprintf(stderr, "Připojení selhalo\n");
```

```
            if (mysql_errno(&my_connection)) {
                fprintf(stderr, "Chyba připojení %d: %s\n",
                    mysql_errno(&my_connection), mysql_error(&my_connection));
        }
    }

    return EXIT_SUCCESS;
}
```

Načtení dat po jednotlivých řádcích

Potřebujeme-li data načítat po jednotlivých řádcích, vyvoláme namísto funkce `mysql_store_result` funkci `mysql_use_result`:

```
MYSQL_RES *mysql_use_result(MYSQL *connection);
```

Podobně jako funkce `mysql_store_result` vrací i funkce `mysql_use_result` při chybě hodnotu `NULL`, při úspěchu pak vrací ukazatel na objekt výsledné množiny. Do výsledné množiny, kterou inicializuje, nenačítá ale žádná data.

> Pro vlastní načtení dat musíme opakovaně volat funkci `mysql_fetch_row`, dokud z výsledné množiny nedostaneme veškerá data. Pokud z množiny navrácené po volání `mysql_use_result` nepřečteme úplně všechna data, mohou další operace pro načítání dat vracet do programu nesprávné informace.

Jaký je tedy rozdíl mezi voláním funkcí `mysql_use_result` a `mysql_store_result`? První z nich pracuje podstatně efektivněji se systémovými prostředky, ale nemůžeme ji použít ve spojení s funkcemi `mysql_data_seek`, `mysql_row_seek` či `mysql_row_tell`, a také funkce `mysql_num_rows` je u ní omezená, protože se dokáže spustit až po načtení veškerých dat.

Zároveň jsme takto zvětšili zpoždění (latenci) aplikace, protože každý požadavek řádku musí projít po síti a zpáteční stejnou cestou i výsledky. Dalším rizikem je, že síťové připojení může selhat uprostřed operace a my se můžeme ocitnout v situaci s neúplnými daty.

Všechny tyto nevýhody však blednou ve srovnání se zřejmými výhodami, kterými je lépe vyvážené zatížení sítě a nižší záznamová režie, zejména u velmi velkých datových množin.

Změnit program `select1.c` na `select2.c`, jenž bude využívat funkci `mysql_use_result`, je velice snadné, takže tu uvádíme jen upravenou část se stínovanými řádky:

```
        if (res) {
            printf("Chyba při SELECT: %s\n", mysql_error(&my_connection));
        } else {
            res_ptr = mysql_use_result(&my_connection);
            if (res_ptr) {
                while ((sqlrow = mysql_fetch_row(res_ptr))) {
                    printf("Načtená data...\n");
                }
                if (mysql_errno(&my_connection)) {
                    printf("Chyba při načítání: %s\n", mysql_error(&my_connection));
                }
                mysql_free_result(res_ptr);
```

```
        }

    }
```

Všimněte si, že až do okamžiku načtení posledního výsledku skutečně nelze zjistit počet dotčených řádků. Při časté a dostatečně včasné kontrole chyb se nám ale s funkcí `mysql_use_result` bude pracovat docela dobře. Takovéto programování nám ušetří spoustu nepříjemné práce při pozdějších úpravách aplikace.

Zpracování navrácených dat

Řádky dat umíme z databáze načítat a můžeme se podívat na jejich vlastní zpracování.

MySQL, stejně jako většina jiných databází SQL, vrací z dotazů dva druhy údajů:

- načtené informace z tabulky, tedy data ze sloupců,
- data o datech neboli *metadata*, tedy například názvy a typy sloupců.

Podívejme se nejprve, jak do čitelné podoby převést samotná data.

Jisté základní informace o výsledcích dotazu zjistíme z funkce `mysql_field_count`, která přebírá objekt připojení a vrací počet polí neboli sloupců ve výsledné množině:

```
unsigned int mysql_field_count(MYSQL *connection);
```

V obecnějším pojetí můžeme funkci `mysql_field_count` využít i pro jiné účely, například zjistit, proč selhalo volání `mysql_store_result`. Jestliže totiž funkce `mysql_store_result` vrátila `NULL`, ale z `mysql_field_count` se vrátilo kladné číslo, znamená to zřejmou chybu při načítání. Pokud ale funkce `mysql_store_result` vrátí nulu, neobsahovala výsledná množina žádné sloupce, a proto ji také nebylo možné uložit. A počet sloupců ve výsledcích dotazu bychom měli znát (dotaz jsme přece napsali sami). Funkce je tedy nejužitečnější v obecných komponentách pro zpracování dotazů nebo v situacích, kdy dotazy konstruujeme za běhu.

> V kódu napsaném pro starší verze MySQL se můžete setkat s funkcí `mysql_num_fields`. Ta přebírá strukturu připojení nebo strukturu výsledné množiny a vrací počet sloupců.

Neuvažujeme-li o formátování výstupu, umíme již data vypsat. Zkusme tedy do programu `select2.c` doplnit jednoduchou funkci `display_row`.

> Poznamenejme, že připojení, výslednou množinu i informace o řádku navrácené z funkce `mysql_fetch_row` jsme pro zjednodušení příkladu uložili do globálních proměnných. V ostrém, produkčním kódu bychom něco takového ale nedoporučovali.

1. Tady je velice jednoduchá rutina pro tisk dat:

```
void display_row() {
    unsigned int field_count;

    field_count = 0;
    while (field_count < mysql_field_count(&my_connection)) {
        printf("%s ", sqlrow[field_count]);
        field_count++;
```

```
    }
    printf("\n");
}
```

2. Funkci doplníme do souboru `select2.c` a přidáme také deklaraci funkce a její volání:

```
void display_row();

int main(int argc, char *argv[]) {
    int res;

    mysql_init(&my_connection);
    if (mysql_real_connect(&my_connection, "localhost", "rick",
                           "bar", "rick", 0, NULL, 0)) {
        printf("Připojení je úspěšné\n");

        res = mysql_query(&my_connection, "SELECT childno, fname,
                          age FROM children WHERE age > 5");
        if (res) {
            printf("SELECT error: %s\n", mysql_error(&my_connection));
       } else {
            res_ptr = mysql_use_result(&my_connection);
            if (res_ptr) {
                while ((sqlrow = mysql_fetch_row(res_ptr))) {
                    printf("Načtená data...\n");
                    display_row();
                }
            }
        }
```

3. Výsledný kód uložíme do souboru `select3.c`, zkompilujeme jej a spustíme:

```
$ gcc -I/usr/include/mysql select3.c -L/usr/lib/mysql -lmysqlclient -o select3
$ ./select3
Připojení je úspěšné
Načtená data...
1 Jenny 21
Načtená data...
2 Andrew 17
$
```

Program tedy funguje, i když výsledky nejsou bůhvíjak estetické. Při výpisu výsledků ale nepočítáme s možností hodnot `NULL`. Chcete-li vypisovat data v lépe formátovaném tvaru (třeba tabulárním), budete kromě dat potřebovat z MySQL i metadata. Pomocí funkce `mysql_fetch_field` je možné současně načíst data i metadata do jediné nové struktury:

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result);
```

Tuto funkci musíme volat opakovaně, dokud nevrátí hodnotu NULL, která indikuje konec dat. Poté můžeme přes ukazatel na strukturu polí přečíst informace o sloupci. Struktura MYSQL_FIELD je definována v hlavičkovém souboru mysql.h, podle následující tabulky:

Pole ve struktuře MYSQL_FIELD	Popis
`char *name;`	Název sloupce jako řetězec.
`char *table;`	Název tabulky, z níž sloupec pochází. Tato položka má smysl především v dotazu prováděném nad mnoha tabulkami. Nezapomeňte také, že u vypočteného sloupce (hodnoty), například MAX, bude název tabulky obsahovat prázdný řetězec.
`char *def;`	Při vyvolání funkce mysql_list_fields (tu zde ale neprobíráme) obsahuje toto pole výchozí hodnotu sloupce.
`enum enum_field_types type;`	Typ sloupce, viz výklad bezprostředně za tabulkou.
`unsigned int length;`	Šířka sloupce, definovaná v okamžiku vytvoření tabulky.
`unsigned int max_length;`	Po zavolání funkce mysql_store_result obsahuje toto pole délku nejdelší načtené hodnoty sloupce (v bajtech); při volání funkce mysql_use_result není přiřazena.
`unsigned int flags;`	Příznaky se vztahují k definici sloupce, nikoli k nalezeným datům. Nejběžnější příznaky (se zřejmým významem) jsou NOT_NULL_FLAG, PRI_KEY_FLAG, UNSIGNED_FLAG, AUTO_INCREMENT_FLAG a BINARY_FLAG. Úplný seznam najdete v dokumentaci k MySQL.
`unsigned int decimals;`	Počet číslic za desetinnou čárkou. Platí jen pro číselná pole.

Různých typů sloupců je docela hodně. Úplný seznam najdete v hlavičkovém souboru mysql_com.h a jak jinak – v dokumentaci. Nejběžnější jsou:

```
FIELD_TYPE_DECIMAL
FIELD_TYPE_LONG
FIELD_TYPE_STRING
FIELD_TYPE_VAR_STRING
```

Jedním zvláště užitečným definovaným makrem je IS_NUM, které vrací hodnotu true pro pole číselného typu, například:

```
if (IS_NUM(myslq_field_ptr->type)) printf("Pole číselného typu\n");
```

Než se pustíme do další aktualizace programu, zmíníme se ještě o jedné funkci:

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result,
                                    MYSQL_FIELD_OFFSET offset);
```

Pomocí této funkce můžete změnit aktuální číslo pole, které se interně inkrementuje při každém volání mysql_fetch_field. Pokud do ní předáme offset rovný nule, vrátíme se na první sloupec.

Máme tedy k dispozici všechny potřebné informace a v programu select můžeme vypsat doplňující údaje k danému sloupci.

Program se bude nazývat `select4.c`; tentokrát uvádíme jeho úplné znění, abyste se mohli podívat na kompletní příklad. Poznamenejme, že se v něm nepokoušíme o nějakou rozsáhlou analýzu typů sloupců; slouží jen pro ověření výše probíraných principů.

```
#include <stdlib.h>
#include <stdio.h>

#include "mysql.h"

MYSQL my_connection;
MYSQL_RES *res_ptr;
MYSQL_ROW sqlrow;

void display_header();
void display_row();

int main(int argc, char *argv[]) {
    int res;
    int first_row = 1; /* Tato proměnná zajistí, že záhlaví řádků
                    vypíšeme při úspěšném načtení dat jen jednou */

    mysql_init(&my_connection);
    if (mysql_real_connect(&my_connection, "localhost", "rick",
                           "tajné", "foo", 0, NULL, 0)) {
        printf("Připojení je úspěšné\n");

        res = mysql_query(&my_connection, "SELECT childno, fname,
                              age FROM children WHERE age > 5");
        if (res) {
           fprintf(stderr, "Chyba při SELECT: %s\n", mysql_error(&my_connection));
       } else {
            res_ptr = mysql_use_result(&my_connection);
            if (res_ptr) {
                while ((sqlrow = mysql_fetch_row(res_ptr))) {
                    if (first_row) {
                        display_header();
                        first_row = 0;
                    }
                    display_row();
                }
                if (mysql_errno(&my_connection)) {
                    fprintf(stderr, "Chyba při načítání: %s\n",
                            mysql_error(&my_connection));
                }
                mysql_free_result(res_ptr);
           }
```

```
        }
        mysql_close(&my_connection);
    } else {
        fprintf(stderr, "Připojení selhalo\n");
        if (mysql_errno(&my_connection)) {
            fprintf(stderr, "Chyba připojení %d: %s\n",
                        mysql_errno(&my_connection),
                        mysql_error(&my_connection));
        }
    }
    return EXIT_SUCCESS;
}

void display_header() {
    MYSQL_FIELD *field_ptr;

    printf("Údaje o sloupcích:\n");
    while ((field_ptr = mysql_fetch_field(res_ptr)) != NULL) {
        printf("\t Název: %s\n", field_ptr->name);
        printf("\t Typ: ");
        if (IS_NUM(field_ptr->type)) {
            printf("Číselné pole\n");
        } else {
            switch(field_ptr->type) {
                case FIELD_TYPE_VAR_STRING:
                    printf("VARCHAR\n");
                    break;
                case FIELD_TYPE_LONG:
                    printf("LONG\n");
                    break;
                default:
                    printf("Typ je %d, zkontrolujte v mysql_com.h\n",
                            field_ptr->type);
            } /* switch */
        } /* else */

        printf("\t Max. šířka %ld\n", field_ptr->length);
        if (field_ptr->flags & AUTO_INCREMENT_FLAG)
            printf("\t Automatická inkrementace\n");
        printf("\n");
    } /* while */
}

void display_row() {
    unsigned int field_count;
```

```
    field_count = 0;
    while (field_count < mysql_field_count(&my_connection)) {
        if (sqlrow[field_count]) printf("%s ", sqlrow[field_count]);
        else printf("NULL");
        field_count++;
   }
    printf("\n");
}
```

Jakmile program zkompilujeme a spustíme, dostaneme takovéto výsledky:

```
$ ./select4
Připojení je úspěšné
Údaje o sloupcích:
    Název: childno
    Typ: Číselné pole
    Max. šířka 11
    Automatická inkrementace

    Název: fname
    Typ: VARCHAR
    Max. šířka 30

    Název: age
    Typ: Číselné pole
    Max. šířka 11
Údaje o sloupcích:
1 Jenny 21
2 Andrew 17
$
```

Výstup z programu ještě není dokonalý, ale pěkně si na něm ilustrujeme možnosti současného zpracování „surových“ dat a metadat, díky nimž můžeme s vlastními daty pracovat efektivněji.

Existují ještě další funkce, které umí například načítat celé pole datových polí a přeskakovat mezi sloupci. Obvykle ale vystačíte jen se základními rutinami, které jsme zde probírali; zájemci najdou potřebné informace v manuálu k MySQL.

Různé funkce

Ještě se podívejte na několik dalších funkcí rozhraní API, které shrnuje následující tabulka. K sestavení funkčního programu by nám měly stačit i dosud probírané funkce, přesto i tato „sbírka“ je docela užitečná.

Příklad volání API	Popis
`char *mysql_get_client_info(void);`	Vrací informace o verzi knihovny, kterou klient používá.
`char *mysql_get_host_info(MYSQL *connection);`	Vrací informace o připojení k serveru.
`char *mysql_get_server_info(MYSQL *connection);`	Vrací informace o serveru, k němuž jsme momentálně připojeni.
`char *mysql_info(MYSQL *connection);`	Vrací informace o naposledy provedeném dotazu, ale funguje jen pro několik typů dotazů, obvykle pro příkazy `INSERT` a `UPDATE`. Pro jiné dotazy vrací `NULL`.
`int mysql_select_db(MYSQL *connection, const char *dbname);`	Změní výchozí databázi na databázi definovanou parametrem, ale za podmínky, že uživatel má příslušná oprávnění. Při úspěchu vrací nulu.
`int mysql_shutdown(MYSQL *connection, enum mysql_enum_shutdown_level shutdown_level);`	Má-li uživatel příslušná oprávnění, ukončí chod databázového serveru, k němuž je momentálně připojen. Do parametru `shutdown_level` je momentálně nutné zapsat hodnotu `SHUTDOWN_DEFAULT`. Při úspěchu vrací nulu.

Aplikace s databází CD

V závěru kapitoly si ukážeme, jak vytvořit jednoduchou databázi pro ukládání informací o sbírce CD a jak k ní přistupovat z programového kódu. Pro zjednodušení vytvoříme jen tři databázové tabulky a mezi nimi nadefinujeme velice jednoduché relace.

Začneme vytvořením nové databáze, kterou pak prohlásíme za aktuální databázi:

```
mysql> create database blpcd;
Query OK, 1 row affected (0.00 sec)

mysql> use blpcd
Connection id: 10
Current database: blpcd

mysql>
```

Nyní v ní můžeme navrhnout a vytvořit potřebné tabulky.

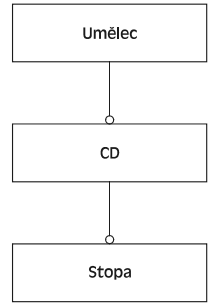
Tento příklad je o něco důmyslnější než předchozí, protože v něm informace o CD rozdělíme do tří samostatných prvků, a sice umělce (nebo skupiny), hlavní katalogové položky a stop. Pokud se zamyslíte nad sbírkou CD a nad tím, z čeho se vlastně skládá, uvědomíte si, že každé CD tvoří řada jednotlivých stop, ale různá CD jsou spolu úzce svázána, například interpretem nebo skupinou, vydavatelskou společností, hudebním stylem a podobně.

Databázi bychom mohli vytvořit hodně složitou a pokusit se všechny tyto tři různé prvky uložit vhodným flexibilním způsobem; v našem příkladu se ale omezíme jen na dvě nejdůležitější relace.

Za prvé, každé CD je tvořeno proměnným počtem stop, takže údaje o stopách budeme zapisovat do samostatné tabulky, oddělené od ostatních dat o CD. Za druhé, jeden umělec či kapela bude

mít často několik alb (CD), takže by bylo rozumné uložit si údaje o umělci jen jednou, protože takto budeme moci načítat všechna CD stejného umělce najednou. Kapely nebudeme rozdělovat na jednotlivé hudebníky, kteří občas vydají sólové album, ani se nebudeme zabývat kompilovanými CD – všechno si zatím zjednodušíme!

Také relace mezi tabulkami budou jednoduché – každý umělec (tím může být i název skupiny) vydal jedno nebo více CD a každé CD se skládá z jedné nebo více stop. Slíbené relace jsou znázorněny na obrázku 8.8.



Obrázek 8.8

Vytvoření tabulek

Dále musíme stanovit vlastní strukturu tabulek. Začneme hlavní tabulkou s CD, kde bude uložena největší část informací. Ke každému CD si musíme uložit ID disku, katalogové číslo, titul a možná ještě nějaké své vlastní poznámky. Navíc potřebujeme číselný identifikátor z tabulky umělců, podle něhož poznáme, který umělec CD vydal.

Tabulka umělců je jednoduchá a bude v ní uloženo vždy pouze jméno a jedinečný ID umělce. Ani tabulka stop není složitá – bude v ní ID disku, k němuž stopa patří, číslo stopy a její titul.

Nejprve tedy tabulka disků CD:

```
CREATE TABLE cd (
    id INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY,
    title VARCHAR(70) NOT NULL,
    artist_id INTEGER NOT NULL,
    catalogue VARCHAR(30) NOT NULL,
    notes VARCHAR(100)
);
```

Tímto příkazem jsme vytvořili tabulku s názvem `cd`, která má následující sloupce:

- celočíselný sloupec `id`, který se automaticky inkrementuje a tvoří primární klíč tabulky,
- znakový sloupec `title` o délce až 70 znaků,
- celočíselný sloupec `artist_id`, který ukazuje do tabulky umělců,
- číselný sloupec `catalogue`, složený nejvýše z 30 znaků,
- až 100 znaků poznámek `notes`.

Všimněte si, že hodnoty NULL jsou povoleny jen ve sloupci notes; ve všech ostatních sloupcích je zadání hodnot povinné.

Přecházíme k tabulce umělců:

```
CREATE TABLE artist (
    id INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

Opět je zde číselný sloupec id a dále jeden sloupec se jménem umělce.

A jako poslední je tu tabulka stop:

```
CREATE TABLE track (
    cd_id INTEGER NOT NULL,
    track_id INTEGER NOT NULL,
    title VARCHAR(70),
    PRIMARY KEY(cd_id, track_id)
);
```

Všimněte si, že v této tabulce definujeme primární klíč jiným způsobem. Tabulka stop je trochu neobvyklá, protože číselný ID každého CD se zde objevuje několikrát, ale stejně tak se několikrát objevuje i číselný ID určité stopy, například stopa číslo 1 – ta by měla být definována u každého CD. Kombinace obou čísel už ale bude vždy jedinečná, takže ji určíme za primární klíč tabulky. Této konstrukci říkáme *složený klíč*, protože se skládá z více než jednoho sloupce.

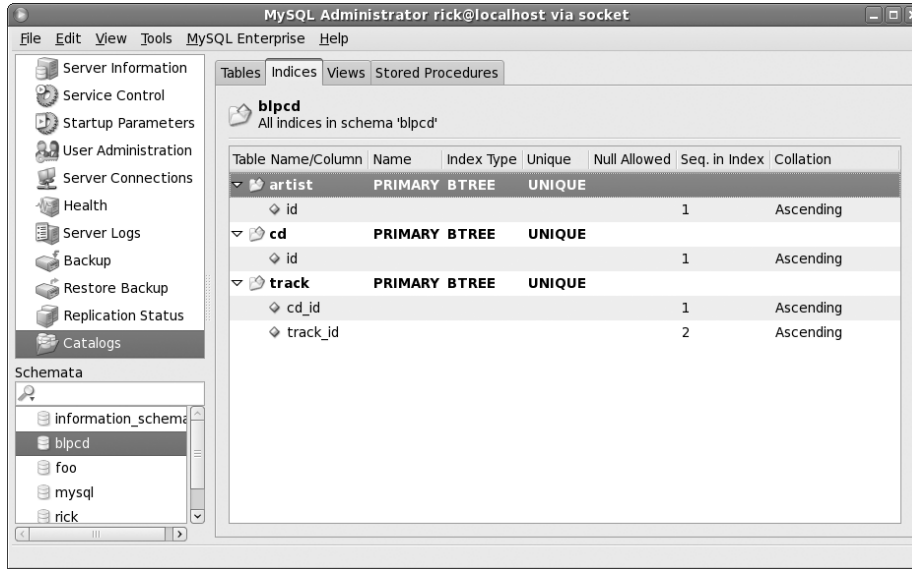
Definiční příkazy SQL uložíme do souboru create_tables.sql, zapíšeme jej do aktuálního adresáře, a poté nejprve vytvoříme databázi a v ní i popsané tabulky. Ukázkový skript z příkladů obsahuje ještě další řádky, které jsou ale zablokovány komentářem; jejich úkolem je odstranit případné existující tabulky.

```
$ mysql -u rick -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> use blpcd;
Database changed
mysql> \. create_tables.sql
Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.00 sec)

mysql>
```

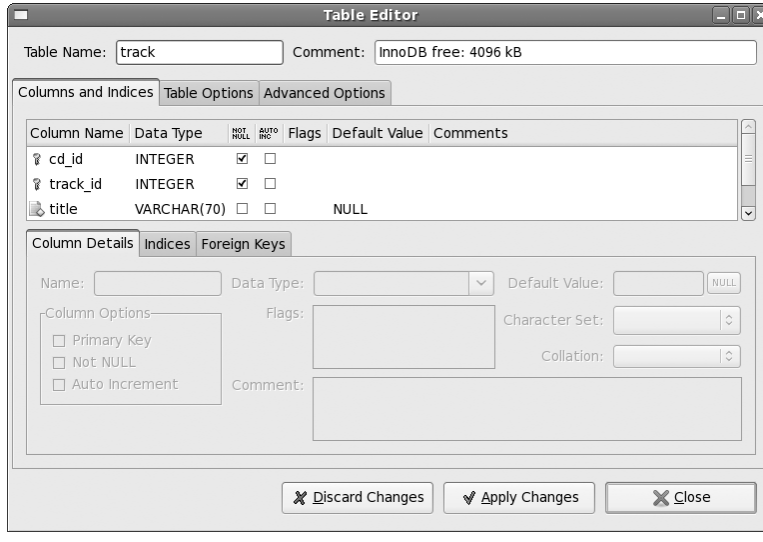
Všimněte si příkazu \., který provádí příkazy načtené ze vstupního souboru – zde create_tables.sql.

Stejně tak bychom tabulky mohli vytvořit spuštěním příkazů SQL anebo přímým zápisem dat do MySQL Query Browseru.

Po vytvoření tabulek si je můžeme prohlédnout v MySQL Administratoru, stejně jako na obrázku 8.9, kde se konkrétně díváme na kartu indexů databáze `blpcd` (tedy na schéma `blpcd`, pokud se vám toto názvosloví víc líbí).



Obrázek 8.9



Obrázek 8.10

Volba pro editaci tabulky (klepněte pravým tlačítkem myši nebo poklepejte na název tabulky v kartě Tables) zobrazí podrobnější informace o sloupcích. Toto okno je na obrázku 8.10.

Všimli jste si na obrázku 8.10 dvou symbolů klíče, a to u polí `cd_id` a `track_id`? Tyto ikony naznačují, že do složeného primárního klíče přispívají oba sloupce. Ve sloupci `track_id` jsme povolili hodnoty `NULL` (volba `NOT NULL` zde není zaškrtnuta), takže počítáme i s neobvyklým, ale ne nemožným případem, kdy stopa na CD nemá žádný titul.

Vložení dat

Tabulky jsou prázdné, takže do nich vložíme data. Nejlepší způsob kontroly jakéhokoli návrhu databáze spočívá ostatně ve vložení vzorových dat.

My si zde ukážeme vzorek importu testových dat; není důležité plně porozumět, co se při importu děje, protože všechny importy jsou v podstatě stejné – jen načítají data do jiných tabulek. Upozorníme jen na dvě důležité věci:

- Skript vymaže všechna existující data, aby tak databáze začínala v čistém stavu.
- Do číselných identifikátorů raději vkládejte přímo zadané hodnoty, než abyste je ponechali na mechanismu `AUTO_INCREMENT`. Tento postup je bezpečnější, protože při jednotlivých operacích vkládání již musíme použít správné hodnoty, které odpovídají definovaným relacím.

Potřebný soubor má název `insert_data.sql` a můžeme jej spustit výše uvedeným příkazem `\.`.

```
--- Smazat stávající data
delete from track;
delete from cd;
delete from artist;


-- Nyní vložit nová data

--- Nejprve do tabulky umělců (nebo skupin)
insert into artist(id, name) values(1, 'Pink Floyd');
insert into artist(id, name) values(2, 'Genesis');
insert into artist(id, name) values(3, 'Einaudi');
insert into artist(id, name) values(4, 'Melanie C');

--- Potom tabulka disků CD
insert into cd(id, title, artist_id, catalogue)
    values(1, 'Dark Side of the Moon', 1, 'B000024D4P');
insert into cd(id, title, artist_id, catalogue)
    values(2, 'Wish You Were Here', 1, 'B000024D4S');
insert into cd(id, title, artist_id, catalogue)
    values(3, 'A Trick of the Tail', 2, 'B000024EXM');
insert into cd(id, title, artist_id, catalogue)
    values(4, 'Selling England By the Pound', 2, 'B000024E9M');
insert into cd(id, title, artist_id, catalogue)
    values(5, 'I Giorni', 3, 'B000071WEV');
```

```
insert into cd(id, title, artist_id, catalogue)
    values(6, 'Northern Star', 4, 'B00004YMST');

--- A naplnění tabulky stop
insert into track(cd_id, track_id, title) values(1, 1, 'Speak to me');
insert into track(cd_id, track_id, title) values(1, 2, 'Breathe');
```

a potom zbytek stop na tomto albu, a potom další album:

```
insert into track(cd_id, track_id, title) values(2, 1, 'Shine on you crazy diamond');
insert into track(cd_id, track_id, title) values(2, 2, 'Welcome to the machine');
insert into track(cd_id, track_id, title) values(2, 3, 'Have a cigar');
insert into track(cd_id, track_id, title) values(2, 4, 'Wish you were here');
insert into track(cd_id, track_id, title)
    values(2, 5, 'Shine on you crazy diamond pt.2');
```

a tak dále...

```
insert into track(cd_id, track_id, title) values(5, 1, 'Melodia Africana (part 1)');
insert into track(cd_id, track_id, title) values(5, 2, 'I due fiumi');
insert into track(cd_id, track_id, title) values(5, 3, 'In un\'altra vita');
```

...až se dostaneme k poslední stopě:

```
insert into track(cd_id, track_id, title) values(6, 11, 'Closer');
insert into track(cd_id, track_id, title) values(6, 12, 'Feel The Sun');
```

Tento skript uložíme pod názvem `pop_tables.sql` a z příkazové výzvy `mysql` jej opět spustíme příkazem `\.`.

> Všimněte si, že stopa 3 na disku 5 (I Giorni) má název „In un'altra vita" s apostrofem uprostřed. Při vkládání dat musíme tento apostrof citovat, tedy jej uvést zpětným lomítkem (`\`).

V tomto okamžiku bychom mohli zkontrolovat, jestli data vypadají rozumně. Opět můžeme využít klienta příkazového řádku a vhodného příkazu SQL. Začneme vybráním prvních dvou stop z každého alba v databázi:

```
SELECT artist.name, cd.title AS "CD Title", track.track_id, track.title AS "Track"
  FROM artist, cd, track
  WHERE artist.id = cd.artist_id AND track.cd_id = cd.id AND track.track_id < 3
```

Vyzkoušíme-li příkaz v MySQL Query Browseru, dostaneme správné výsledky podle obrázku 8.11.

Kód příkazu SQL vypadá na pohled složitě, ale po částech mu porozumíme docela snadno.

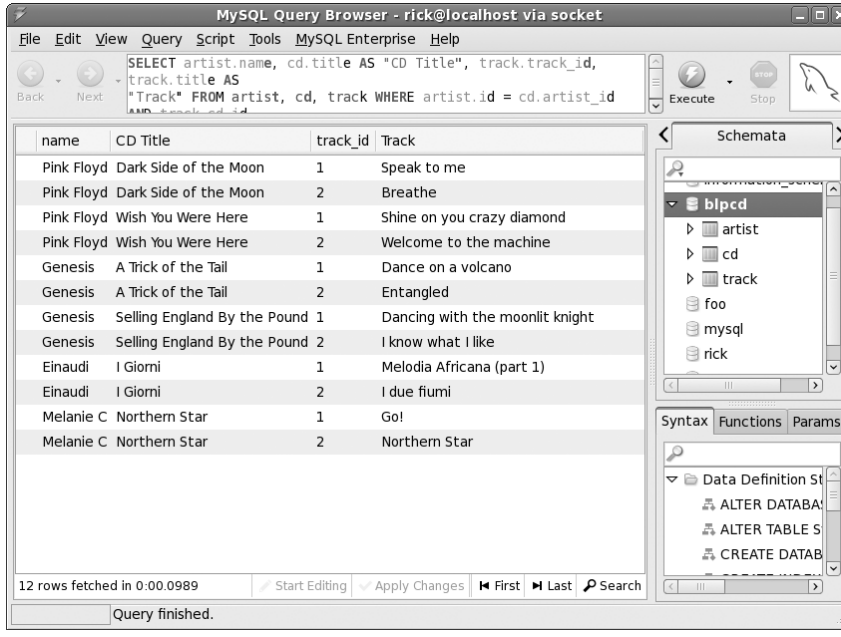
Nejprve v příkazu `SELECT` na chvilku ignorujeme klauzule `AS` a vidíme, že první část příkazu je:

```
SELECT artist.name, cd.title, track.track_id, track.title
```

Tímto zápisem jednoduše říkáme, které sloupce požadujeme zobrazit; zadáváme je vždy ve tvaru `názevtabulky.názevsloupce`.

Klauzule `AS` v příkazu `SELECT`, tedy `cd.title AS "CD Title"` a `track.title AS "Track"`, vypisují do výsledků přejmenované sloupce. Nad sloupcem `title` z tabulky `cd` (`cd.title AS`) bude tedy vy-

psáno záhlaví „CD Title“ a nad sloupcem `track.title` bude text „Track“. Díky klauzuli `AS` jsou tak výsledky čitelnější. Při volání příkazu SQL z jiného jazyka nemá cenu tyto zástupné názvy používat téměř nikdy, ale při práci na příkazovém řádku je klauzule velice užitečná.



Obrázek 8.11

Ani další část příkazu není nijak složitá; server se z ní dozví, s jakými tabulkami má pracovat:

```
FROM artist, cd, track
```

Klauzule `WHERE` je přece jen o něco složitější:

```
WHERE artist.id = cd.artist_id AND track.cd_id = cd.id AND track.track_id < 3
```

První část serveru říká, že sloupec `id` v tabulce umělců má být roven sloupci `artist_id` z tabulky `cd`. Nezapomeňte, že jméno umělce jsme do databáze uložili jen jednou a že z tabulky CD se na ně odvoláváme přes číslo. Další část, `track.cd_id = cd.id`, požaduje podobnou shodu nad tabulkami `track` a `cd`; přesněji požaduje, aby se sloupec `cd_id` v tabulce `track` shodoval se sloupcem `id` z tabulky `cd`. Třetí část podmínky, `track.track_id < 3`, pak zredukuje objem navrácených dat, takže z každého CD povolí výstup jen stopám 1 a 2. A nakonec všechny tři podmínky spojíme logickým součinem `AND`, protože požadujeme splnění všech tří současně.

Přístup k aplikačním datům z jazyka C

V této kapitole ještě nebudeme psát úplnou aplikaci s grafickým uživatelským rozhraním; naším úkolem bude napsat jen kód rozhraní, které bude rozumně jednoduchým způsobem přistupovat k datům z programu v C. Obvyklým problémem při psaní podobného kódu bývá neznámý počet navrácených výsledků a vhodný způsob jejich předávání mezi klientským kódem a kódem pro

přístup k databázi. V této aplikaci se chceme zaměřit na databázové rozhraní a chceme si všechno zjednodušit, a proto budeme pracovat se strukturami pevné velikosti. Pro skutečnou aplikaci by ale takovýto postup byl nepřijatelný. Běžným řešením, které zároveň pomáhá snížit objem síťového provozu, je načítání dat po jednotlivých řádcích, jak jsme již v této kapitole viděli v souvislosti s funkcemi `mysql_use_result` a `mysql_fetch_row`.

Definice rozhraní

Začneme hlavičkovým souborem, v němž nadefinujeme potřebné struktury a funkce a který pojmenujeme `app_mysql.h`.

Nejprve zmíněné struktury:

```
/* Zjednodušená struktura pro reprezentaci aktuálního CD,
bez informací o stopách */
struct current_cd_st {
   int artist_id;
   int cd_id;
   char artist_name[100];
   char title[100];
   char catalogue[100];
};

/* Zjednodušená struktura informací o stopě */
struct current_tracks_st {
   int cd_id;
   char track[20][100];
};

#define MAX_CD_RESULT 10
struct cd_search_st {
   int cd_id[MAX_CD_RESULT];
};
```

Dále je tu dvojice funkcí pro připojení a odpojení od databáze:

```
/* Funkce pro obsluhu databáze na pozadí */
int database_start(char *name, char *password);
void database_end();
```

Nyní přejdeme k vlastním funkcím pro manipulaci s daty. Všimněte si, že tu chybí zvláštní funkce pro vytváření a odstraňování umělců; ty budeme implementovat jen v „zákulisí“ aplikace a položky umělců budeme vytvářet vždy podle potřeby a odstraňovat je, až se nebudou používat v žádném albu.

```
/* Funkce pro přidání CD */
int add_cd(char *artist, char *title, char *catalogue, int *cd_id);
int add_tracks(struct current_tracks_st *tracks);
/* Funkce pro vyhledávání a načítání CD */
int find_cds(char *search_str, struct cd_search_st *results);
```

```
int get_cd(int cd_id, struct current_cd_st *dest);
int get_cd_tracks(int cd_id, struct current_tracks_st *dest);
/* Funkce pro odstraňování položek */
int delete_cd(int cd_id);
```

Funkce pro vyhledávání je zcela obecná: předáme do ní řetězec, který nám vyhledá mezi položkami umělců, titulů i katalogu.

Testování aplikačního rozhraní

Před implementací samotného rozhraní musíme ještě napsat nějaký kód, kterým je vyzkoušíme. Na pohled se to může zdát zbytečné, ale takto si velice dobře uvědomíme, jak přesně se rozhraní má chovat, a můžeme se tak pustit do jeho správné implementace.

Kód uložíme do souboru `app_test.c`. Nejprve pár deklarací `#include` a struktur:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "app_mysql.h"

int main() {
    struct current_cd_st cd;
    struct cd_search_st cd_res;
    struct current_tracks_st ct;
    int cd_id;
    int res, i;
```

Jako první musíme v aplikaci vždy inicializovat databázové připojení a zadat do něj platné uživatelské jméno a heslo (to musí samozřejmě platit na vašem konkrétním systému):

```
    database_start("rick", "tajné");
```

Poté vyzkoušíme přidání nového CD:

```
    res = add_cd("Mahler", "Symphony No 1", "4596102", &cd_id);
    printf("Výsledek přidání CD byl %d, cd_id je %d\n", res, cd_id);

    memset(&ct, 0, sizeof(ct));
    ct.cd_id = cd_id;
    strcpy(ct.track[0], "Langsam Schleppend");
    strcpy(ct.track[1], "Kraftig bewegt");
    strcpy(ct.track[2], "Feierlich und gemessen");
    strcpy(ct.track[3], "Sturmisch bewegt");
    add_tracks(&ct);
```

Nyní toto CD vyhledáme a načteme informace z prvního nalezeného CD:

```
    res = find_cds("Symphony", &cd_res);
    printf("Nalezeno %d CD, první má ID %d\n", res, cd_res.cd_id[0]);
```

```
    res = get_cd(cd_res.cd_id[0], &cd);
    printf("get_cd vrátilo %d\n", res);

    memset(&ct, 0, sizeof(ct));
    res = get_cd_tracks(cd_res.cd_id[0], &ct);
    printf("get_cd_tracks vrátilo %d\n", res);
    printf("Titul: %s\n", cd.title);

    i = 0;
    while (i < res) {
        printf("\tstopa %d je %s\n", i, ct.track[i]);
        i++;
   }
```

Nakonec CD odstraníme:

```
    res = delete_cd(cd_res.cd_id[0]);
    printf("Delete_cd vrátilo %d\n", res);
```

A ještě se odpojíme od databáze a skončíme program:

```
    database_end();

    return EXIT_SUCCESS;
}
```

Implementace rozhraní

Dostáváme se ke složitější části a tou je implementace navrženého rozhraní. Celé rozhraní zapíšeme do souboru `app_mysql.c`.

Začneme opět základními direktivami `#include`, globální strukturou připojení a dále příznakem `dbconnected`, který zajistí, že se aplikace nebudou pokoušet o přístup k datům bez navázaného připojení. Pro zlepšení struktury kódu zavádíme interní funkci `get_artist_id`.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "mysql.h"
#include "app_mysql.h"

static MYSQL my_connection;
static int dbconnected = 0;

static int get_artist_id(char *artist);
```

Navázat připojení k databázi je velice jednoduché, jak jsme si už v kapitole ukázali, a odpojení od databáze je ještě jednodušší:

```
int database_start(char *name, char *pwd) {
    if (dbconnected) return 1;

    mysql_init(&my_connection);
    if (!mysql_real_connect(&my_connection, "localhost", name, pwd,
            "blpcd", 0, NULL, 0)) {
        fprintf(stderr, "Připojení k databázi selhalo: %d, %s\n",
        mysql_errno(&my_connection), mysql_error(&my_connection));
        return 0;
   }
    dbconnected = 1;
    return 1;
} /* database start */

void database_end() {
    if (dbconnected) mysql_close(&my_connection);
    dbconnected = 0;
} /* database_end */
```

Teď začne skutečná práce, a to ve funkci `add_cd`. Nejprve přichází několik deklarací a několik kontrol, při nichž si ověříme, jestli jsme správně připojeni k databázi. Tato kontrola nechybí v žádné naší zvenku dostupné funkci.

```
int add_cd(char *artist, char *title, char *catalogue, int *cd_id) {
    MYSQL_RES *res_ptr;
    MYSQL_ROW mysqlrow;
    int res;
    char is[250];
    char es[250];
    int artist_id = -1;
    int new_cd_id = -1;
    if (!dbconnected) return 0;
```

Jako další věc musíme zkontrolovat, jestli požadovaný umělec existuje, a v opačném případě jej vytvořit. Všechny tyto operace provádí funkce `get_artist_id`, jejíž implementace následuje za chvíli.

```
    artist_id = get_artist_id(artist);
```

Nyní tedy máme kód `artist_id` a můžeme vložit hlavní záznam CD. Všimněte si volání funkce `mysql_escape_string`, která slouží k zachování případných speciálních znaků v titulu CD:

```
    mysql_escape_string(es, title, strlen(title));
    sprintf(is, "INSERT INTO cd(title, artist_id, catalogue)
            VALUES('%s', %d, '%s')", es, artist_id, catalogue);
    res = mysql_query(&my_connection, is);
    if (res) {
        fprintf(stderr, "Chyba při vložení %d: %s\n",
        mysql_errno(&my_connection), mysql_error(&my_connection));
        return 0;
   }
```

Jakmile se dostaneme k přidávání stop na nové CD, musíme znát jeho číselný ID, vytvořený v okamžiku vložení. Příslušné pole má ale automatickou inkrementaci, takže databáze jej přiřadila automaticky – my je nyní potřebujeme načíst explicitně a využijeme tedy speciální funkci `LAST_INSERT_ID`, s níž jsme se v kapitole také seznámili:

```
    res = mysql_query(&my_connection, "SELECT LAST_INSERT_ID()");
    if (res) {
        printf("Chyba při SELECT: %s\n", mysql_error(&my_connection));
        return 0;
   } else {
        res_ptr = mysql_use_result(&my_connection);
        if (res_ptr) {
            if ((mysqlrow = mysql_fetch_row(res_ptr))) {
                sscanf(mysqlrow[0], "%d", &new_cd_id);
            }
            mysql_free_result(res_ptr);
       }
```

Možností, že současně vkládají do databáze nějaká CD i jiní klienti a že tak dojde ke zmatkům v číselných ID, se zabývat nemusíme, protože MySQL sleduje přiřazené ID ke každému klientskému připojení zvlášť. I když tedy jiná aplikace vloží další CD ještě dříve, než načteme náš vlastní ID, dostaneme z něj správnou hodnotu, nikoli hodnotu zapsanou z této druhé aplikace.

Nakonec přiřadíme ID nově přidaného řádku a vrátíme kód úspěchu nebo selhání:

```
        *cd_id = new_cd_id;
        if (new_cd_id!= -1) return 1;
        return 0;
   }
} /* add_cd */
```

Nyní se podíváme na implementaci funkce `get_artist_id`; postup je velice podobný vkládání záznamu s CD:

```
/* Najde nebo vytvoří artist_id pro zadaný řetězec */
static int get_artist_id(char *artist) {
    MYSQL_RES *res_ptr;
    MYSQL_ROW mysqlrow;

    int res;
    char qs[250];
    char is[250];
    char es[250];
    int artist_id = -1;

    /* Existuje již umělec? */
    mysql_escape_string(es, artist, strlen(artist));
    sprintf(qs, "SELECT id FROM artist WHERE name = '%s'", es);
```

```
    res = mysql_query(&my_connection, qs);
    if (res) {
        fprintf(stderr, "Chyba SELECT: %s\n", mysql_error(&my_connection));
   } else {
        res_ptr = mysql_store_result(&my_connection);
        if (res_ptr) {
            if (mysql_num_rows(res_ptr) > 0) {
                if (mysqlrow = mysql_fetch_row(res_ptr)) {
                    sscanf(mysqlrow[0], "%d", &artist_id);
                }
            }
            mysql_free_result(res_ptr);
        }
   }
    if (artist_id!= -1) return artist_id;

    sprintf(is, "INSERT INTO artist(name) VALUES('%s')", es);
    res = mysql_query(&my_connection, is);
    if (res) {
        fprintf(stderr, "Chyba při vložení %d: %s\n",
        mysql_errno(&my_connection), mysql_error(&my_connection));
        return 0;
   }

    res = mysql_query(&my_connection, "SELECT LAST_INSERT_ID()");
    if (res) {
        printf("Chyba při SELECT: %s\n", mysql_error(&my_connection));
        return 0;
   } else {
        res_ptr = mysql_use_result(&my_connection);
        if (res_ptr) {
            if ((mysqlrow = mysql_fetch_row(res_ptr))) {
                sscanf(mysqlrow[0], "%d", &artist_id);
            }
            mysql_free_result(res_ptr);
        }
   }
    return artist_id;
} /* get_artist_id */
```

Nyní si ukážeme přidání informací o stopách na novém CD. Opět musíme zachovat případné speciální znaky v titulu:

```
int add_tracks(struct current_tracks_st *tracks) {
    int res;
    char is[250];
    char es[250];
```

```
    int i;

    if (!dbconnected) return 0;

    i = 0;
    while (tracks->track[i][0]) {
        mysql_escape_string(es, tracks->track[i], strlen(tracks->track[i]));
        sprintf(is, "INSERT INTO track(cd_id, track_id, title)
        VALUES(%d, %d, '%s')", tracks->cd_id, i + 1, es);
        res = mysql_query(&my_connection, is);
        if (res) {
            fprintf(stderr, "Chyba při vložení %d: %s\n",
            mysql_errno(&my_connection), mysql_error(&my_connection));
            return 0;
       }
        i++;
   }
    return 1;
} /* add_tracks */
```

V další části přecházíme k načítání informací o CD podle zadané hodnoty ID. Zde budeme potřebovat databázové spojení tabulek (join), kdy společně s informacemi o CD načteme také údaje o umělci. Tento postup je velice rozumný, protože databáze umí efektivně provádět i složité dotazy. Nepokoušejte se proto v kódu aplikace dělat věci, které se dají jednoduše zapsat v jazyce SQL; tím si nejen ušetříte zbytečnou práci, ale zároveň urychlíte činnost aplikace, která podstatnou část práce „svěří“ databázovému stroji.

```
int get_cd(int cd_id, struct current_cd_st *dest) {
    MYSQL_RES *res_ptr;
    MYSQL_ROW mysqlrow;

    int res;
    char qs[250];

    if (!dbconnected) return 0;
    memset(dest, 0, sizeof(*dest));
    dest->artist_id = -1;

    sprintf(qs, "SELECT artist.id, cd.id, artist.name, cd.title, cd.catalogue \
    FROM artist, cd WHERE artist.id = cd.artist_id and cd.id = %d", cd_id);

    res = mysql_query(&my_connection, qs);
    if (res) {
        fprintf(stderr, "Chyba při SELECT: %s\n", mysql_error(&my_connection));
   } else {
        res_ptr = mysql_store_result(&my_connection);
        if (res_ptr) {
```

```
            if (mysql_num_rows(res_ptr) > 0) {
                if (mysqlrow = mysql_fetch_row(res_ptr)) {
                    sscanf(mysqlrow[0], "%d", &dest->artist_id);
                    sscanf(mysqlrow[1], "%d", &dest->cd_id);
                    strcpy(dest->artist_name, mysqlrow[2]);
                    strcpy(dest->title, mysqlrow[3]);
                    strcpy(dest->catalogue, mysqlrow[4]);
                }
            }
            mysql_free_result(res_ptr);
        }
    }
    if (dest->artist_id!= -1) return 1;
    return 0;
} /* get_cd */
```

Dále přichází na řadu implementace funkce pro načítání informací o stopách. Všimněte si, že jsme do příkazu SQL zapsali klauzuli `ORDER BY`, která vrací stopy v „rozumném“ pořadí. Tuto operaci opět ponecháme na databázi, která ji zvládne mnohem efektivněji než my sami v programovém kódu:

```
int get_cd_tracks(int cd_id, struct current_tracks_st *dest) {
    MYSQL_RES *res_ptr;
    MYSQL_ROW mysqlrow;

    int res;
    char qs[250];
    int i = 0, num_tracks = 0;

    if (!dbconnected) return 0;
    memset(dest, 0, sizeof(*dest));
    dest->cd_id = -1;

    sprintf(qs, "SELECT track_id, title FROM track WHERE track.cd_id = %d \
            ORDER BY track_id", cd_id);

    res = mysql_query(&my_connection, qs);
    if (res) {
        fprintf(stderr, "Chyba při SELECT: %s\n", mysql_error(&my_connection));
    } else {
        res_ptr = mysql_store_result(&my_connection);
        if (res_ptr) {
            if ((num_tracks = mysql_num_rows(res_ptr)) > 0) {
                while (mysqlrow = mysql_fetch_row(res_ptr)) {
                    strcpy(dest->track[i], mysqlrow[1]);
                    i++;
                }
                dest->cd_id = cd_id;
```

```
            }
             mysql_free_result(res_ptr);
        }
    }
     return num_tracks;
} /* get_cd_tracks */
```

Zatím jsme zvládli funkce pro vkládání a načítání informací o CD a nyní je na čase podívat se na vyhledávání. Pro jednoduchost rozhraní jsme omezili počet navrácených výsledků, zároveň ale chceme zjistit, kolik řádků bylo nalezeno (přestože je jich více, než kolik najednou načteme).

```
int find_cds(char *search_str, struct cd_search_st *dest) {
    MYSQL_RES *res_ptr;
    MYSQL_ROW mysqlrow;

    int res;
    char qs[500];
    int i = 0;
    char ss[250];
    int num_rows = 0;

    if (!dbconnected) return 0;
```

Vymažeme strukturu výsledků a opět nezapomeneme na zachování speciálních znaků v dotazu:

```
    memset(dest, -1, sizeof(*dest));
    mysql_escape_string(ss, search_str, strlen(search_str));
```

Dále musíme sestavit řetězec s dotazem. Všimněte si velkého množství znaků %; to je proto, že znak procenta potřebujeme jednak zapsat do dotazu SQL (kde se shoduje s libovolným řetězcem) a jednak je speciálním znakem ve funkci `sprintf`.

```
    sprintf(qs, "SELECT DISTINCT artist.id, cd.id FROM artist, cd WHERE artist.id =
            cd.artist_id and (artist.name LIKE '%%%s%%' OR cd.title LIKE '%%%s%%' OR
            cd.catalogue LIKE '%%%s%%')", ss, ss, ss);
```

Hotový dotaz můžeme spustit:

```
    res = mysql_query(&my_connection, qs);

    if (res) {
        fprintf(stderr, "Chyba při SELECT: %s\n", mysql_error(&my_connection));
    } else {
        res_ptr = mysql_store_result(&my_connection);
        if (res_ptr) {
            num_rows = mysql_num_rows(res_ptr);
            if (num_rows > 0) {
                while ((mysqlrow = mysql_fetch_row(res_ptr)) && i < MAX_CD_RESULT) {
                    sscanf(mysqlrow[1], "%d", &dest->cd_id[i]);
                    i++;
```

```
            }
          }
           mysql_free_result(res_ptr);
       }
   }
    return num_rows;
} /* find_cds */
```

A nakonec se pustíme do implementace odstraňování CD. V souladu s politikou „tiché“ obsluhy položek umělců musíme odstranit umělce daného CD, pokud už nemá v databázi žádné jiné CD. Jazyk SQL kupodivu neumí vyjádřit odstranění dat z několika tabulek, takže musíme z každé tabulky odstranit řádky zvlášť.

```
int delete_cd(int cd_id) {
    int res;
    char qs[250];
    int artist_id, num_rows;
    MYSQL_RES *res_ptr;
    MYSQL_ROW mysqlrow;

    if (!dbconnected) return 0;

    artist_id = -1;
    sprintf(qs, "SELECT artist_id FROM cd WHERE artist_id =
            (SELECT artist_id FROM cd WHERE id = '%d')", cd_id);
    res = mysql_query(&my_connection, qs);

    if (res) {
        fprintf(stderr, "Chyba při SELECT: %s\n", mysql_error(&my_connection));
   } else {
        res_ptr = mysql_store_result(&my_connection);
        if (res_ptr) {
            num_rows = mysql_num_rows(res_ptr);
            if (num_rows == 1) {
                /* Umělec se nevyskytuje na žádném jiném CD */
                mysqlrow = mysql_fetch_row(res_ptr);
                sscanf(mysqlrow[0], "%d", &artist_id);
            }
        mysql_free_result(res_ptr);
       }
   }

    sprintf(qs, "DELETE FROM track WHERE cd_id = '%d'", cd_id);
    res = mysql_query(&my_connection, qs);
    if (res) {
        fprintf(stderr, "Chyba při odstranění stopy %d: %s\n",
        mysql_errno(&my_connection), mysql_error(&my_connection));
```

```
        return 0;
    }

    sprintf(qs, "DELETE FROM cd WHERE id = '%d'", cd_id);
    res = mysql_query(&my_connection, qs);
    if (res) {
        fprintf(stderr, "Chyba při odstranění CD %d: %s\n",
        mysql_errno(&my_connection), mysql_error(&my_connection));
        return 0;
    }

    if (artist_id!= -1) {
        /* k položce tohoto umělce neexistuje žádné jiné CD, smažeme jej */
        sprintf(qs, "DELETE FROM artist WHERE id = '%d'", artist_id);
        res = mysql_query(&my_connection, qs);
        if (res) {
            fprintf(stderr, "Chyba při odstranění umělce %d: %s\n",
            mysql_errno(&my_connection), mysql_error(&my_connection));
        }
    }
    return 1;
} /* delete_cd */
```

Tím je kód příkladu hotový.

Pro úplnost napíšeme ještě soubor `Makefile`, který nám usnadní aplikaci. Podle podmínek konkrétní instalace MySQL v systému budete možná muset změnit cestu k souborům include:

```
all: app

app: app_mysql.c app_test.c app_mysql.h
    gcc -o app -I/usr/include/mysql app_mysql.c app_test.c
                                -lmysqlclient -L/usr/lib/mysql
```

V dalších kapitolách na toto programové rozhraní napojíme plnohodnotné grafické uživatelské rozhraní (GUI). Chcete-li i v této podobě sledovat, jak se databáze za chodu programu mění, můžete kód krokovat v debuggeru `gdb` a z druhého okna sledovat změny dat. A pokud pracujete s nástrojem MySQL Query Browser, nezapomeňte, že po každém kroku změn musíte aktualizovat zobrazení dat.

Shrnutí

Osmá kapitola knížky nás stručně seznámila s databázovým strojem MySQL. Zkušenější uživatelé se jistě budou zajímat o řadu pokročilých funkcí, na jejichž výklad jsme v textu neměli čas, například o omezení cizích klíčů nebo spouště (triggers).

Řekli jsme si základy instalace MySQL a podívali jsme se na základní administraci databází MySQL z klientských utilit. Ukázali jsme si také rozhraní API pro C, který patří mezi jazyky, z nichž lze databáze MySQL ovládat. A naučili jsme se něco málo z jazyka SQL.

Doufáme, že ve vás tato kapitola vzbudila zájem o správu dat v databázích SQL a také o další studium těchto výkonných nástrojů pro správu databází.

A ještě jednou připomínáme, že hlavním zdrojem informací o MySQL je domovská stránka MySQL na adrese `www.mysql.com`.