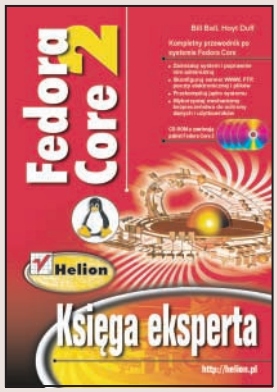
Fedora Core 2. Księga eksperta



Autor: Bill Ball, Hoyt Duff
Tłumaczenie: Grzegorz Kowalczyk (wstęp, rozdz. 1 – 11, 14, 15), Przemysław Szeremiota (rozdz. 16 – 27, dod. A – C)
ISBN: 83-7361-553-9
Tytuł oryginału: Red Hat Linux and Fedora Unleashed
Format: B5, stron: 1080

Kompletny przewodnik po systemie Fedora Core

- Zainstaluj system i poprawnie nim administruj
- Skonfiguruj serwer WWW, FTP, poczty elektronicznej i plików
- Przekompiluj jądro systemu
- Wykorzystaj mechanizmy bezpieczeństwa do ochrony danych i użytkowników

Fedora Core to najnowsza, bezpłatna i otwarta kontynuacja linii systemu Red Hat Linux, najpopularniejszej dystrybucji systemu Linux na świecie. Jej poprzednik – Red Hat Linux – zyskał zasłużoną sławę i popularność nie tylko wśród komputerowych guru, ale także „zwykłych" użytkowników. Docenili jego stabilność, łatwość obsługi i ogromny wybór różnego rodzaju aplikacji, dostępnych nieodpłatnie, podobnie jak sam system. Fedora Core posiada te same zalety. Można zainstalować ten system nie tylko na komputerze PC, ale także na serwerach sieciowych, urządzeniach PDA i komputerach klasy mainframe.

„Fedora Core 2. Księga eksperta" to kompendium wiedzy na temat systemu Fedora Core 2 Linux. Zawiera szczegółowe informacje na temat jego instalacji, używania i administrowania nim. Opisuje sposoby konfigurowania systemu oraz korzystania z niego zarówno na stacjach roboczych, jak i na serwerach sieciowych. Przedstawia zasady używania środowiska tekstowego i graficznego, konfigurowania usług systemowych, zarządzania kontami użytkowników i wiele innych informacji cennych zarówno dla użytkownika, jak i dla administratora.

- Instalacja i konfigurowanie systemu
- System plików, logowanie i przydzielanie praw użytkownikom
- Korzystanie ze środowiska X Window
- Zarządzanie usługami systemowymi
- Instalowanie oprogramowania
- Administrowanie kontami użytkowników i systemem plików
- Tworzenie kopii bezpieczeństwa
- Konfiguracja sieci i drukarek
- Instalacja i konfigurowanie serwera DNS, WWW, FTP i poczty elektronicznej
- Usługi baz danych
- Tworzenie skryptów powłoki oraz oprogramowania w językach C++ oraz Perl
- Konfigurowanie jądra systemu
- Aplikacje biurowe, multimedialne i emulatory innych systemów operacyjnych

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 17.
Administrowanie usługami baz danych

Bieżący rozdział stanowić będzie wprowadzenie do korzystania z baz danych MySQL i PostgreSQL, dwóch systemów baz danych rozprowadzanych wraz z dystrybucją Fedora Core 2. Czytelnik będzie miał okazję poznać podstawowe funkcje tych systemów, porównać ich możliwości i poznać zalety i wady obu. Informacje te pomocne będą przy dokonywaniu wyboru systemu, którego zadaniem będzie obsługa baz danych danej organizacji. Omówienie zawierać będzie również opis procedur instalacji, inicjalizacji i konfiguracji tych popularnych serwerów baz danych, jak również sposobu administrowania ich działaniem i utrzymywania bezpieczeństwa i spójności danych zarządzanych przez te serwery.

Administrator bazy danych ma w organizacji kilka zadań, zależnych w dużej mierze od rozmiaru i zakresu działania organizacji, liczebności zespołu działu informatycznego i tak dalej. W zależności od struktury danej organizacji zakres odpowiedzialności administratora bazy danych może obejmować:

- Instalowanie i konserwacja **serwerów baz danych** — konserwacja oznacza tu zadanie mające na celu podtrzymanie dostępności danych i dbanie o stabilność serwera, a więc na przykład instalowanie publikowanych łat eliminujących błędy i luki w zabezpieczeniach czy aktualizowanie oprogramowania. Administrator bazy danych potrzebuje do realizacji tych zadań uprawnień użytkownika uprzywilejowanego oraz umiejętności zarządzania oprogramowaniem (patrz rozdział 8., „Zarządzanie oprogramowaniem i zasobami systemu"). Nie zaszkodzi też znajomość jądra systemu, wykorzystywanych systemów plików i innych elementów, mających wpływ czy to na wydajność, czy też bezpieczeństwo dostępu do danych.
- Instalowanie i konserwacja oprogramowania **klientów baz danych** — klient bazy danych to program, który w imieniu użytkownika nawiązuje połączenie z serwerem bazy danych i udostępnia użytkownikowi otrzymane dane (patrz podrozdział „Klienty baz danych"), czy to odwołując się do serwera lokalnego, czy też nawiązując połączenie za pośrednictwem sieci komputerowej. Odpowiedzialność administratora w tym zakresie może obejmować instalowanie i konserwację oprogramowania klienckiego. Rozdział ten prezentuje między innymi sposób instalowania takiego oprogramowania oraz korzystania z oprogramowania klienckiego, z poziomu wiersza poleceń oraz za pośrednictwem interfejsu graficznego.

- Zarządzanie kontami i użytkownikami — zarządzanie kontami i użytkownikami obejmuje dodawanie i usuwanie użytkowników bazy danych, przypisywanie haseł i zarządzanie hasłami i tak dalej. W rozdziale pokazana zostanie między innymi procedura przyznawania i odbierania użytkownikowi bazy danych uprawnień dostępu do danych i przypisywania hasła (prezentacja obejmować będzie oba systemy baz danych).
- Zapewnianie bezpieczeństwa bazy danych — w tym zakresie należy zadbać o takie elementy, jak właściwa kontrola dostępu, blokująca dostęp do danych osobom nieupoważnionym, oraz system uprawnień, pozwalający na regulowanie zakresu działalności użytkowników bazy danych. W rozdziale pokazany zostanie sposób zarządzania dostępem do bazy danych za pośrednictwem WWW, klienta protokołu SSH oraz lokalnego klienta wyposażonego w interfejs graficzny. Bezpieczeństwo danych jest również silnie uzależnione od starannego zaplanowania a następnie konsekwentnego wdrażania harmonogramu wykonywania kopii zapasowych danych.
- Zapewnianie spójności danych — z wszystkich informacji przechowywanych na dysku serwera najważniejsze mogą okazać się te przechowywane w bazie danych. Zapewnienie spójności tych danych obejmuje regulowanie współbieżnego dostępu do danych i zapewnianie, że żadne z modyfikacji wprowadzonych w tym samym czasie przez różnych użytkowników nie zostaną ani utracone, ani nie będą się dublować.

Krótkie wprowadzenie do baz danych

Usługi baz danych w systemie Linux, wykorzystujące oprogramowanie omawiane w rozdziale, to usługi oparte na **architekturze klient-serwer**. Klienty bazy danych to programy pośredniczące we wprowadzaniu danych oraz inicjowaniu zapytań do serwera danych, zwracających wyznaczone kryteriami zapytania zbiory danych. Do uruchomionego serwera można się odwołać zarówno za pośrednictwem klienta działającego w oknie terminala, jak i dysponującego interfejsem graficznym. Bazy danych można przy tym podzielić na bazy plikowe i relacyjne. **Plikowa baza danych** może być przechowywana w zwykłym pliku, w którym poszczególne pola i rekordy są oddzielane znakami spacji czy nowego wiersza. Przykładem plikowej bazy danych jest systemowy plik haseł */etc/passwd*. Innym przykładem mógłby być plik prostej książki adresowej, zawierającej następujące wpisy:

```
Doe~John~505 Some Street~Anytown~NY~12345~555-555-1212
```

Do przeszukiwania i wyodrębniania informacji z utworzonej tak prymitywnej bazy danych można wykorzystywać popularne narzędzia, takie jak `grep`, `awk` czy programy w języku Perl. Programy te mogą się zresztą świetnie sprawdzać, dopóki baza danych nie będzie zbyt obszerna; w ogólności jednak plikowe bazy danych są obarczone następującymi ograniczeniami:

- Nie dają się dobrze skalować. Plikowe bazy danych nie mogą realizować swobodnego dostępu do danych. Przeszukiwanie zawsze odbywa się sekwencyjnie. Oznacza to, że wyszukiwanie pewnych informacji jest realizowane przez odczytywanie i analizowanie kolejnych wierszy pliku. W miarę wzrostu rozmiaru bazy danych metoda ta przestaje jednak być efektywna.

- Plikowe bazy danych nie nadają się do wykorzystania w środowisku wielodostępnym. Taka baza danych, w zależności od konfiguracji, może albo zezwalać na modyfikację danych tylko jednemu użytkownikowi naraz, albo — kiedy modyfikacje będą przeprowadzane równocześnie przez kilku użytkowników — dopuszczać potencjalną możliwość utraty i nadpisania danych w wyniku dostępu współbieżnego.

Powyższe ograniczenia czynią plikowe bazy danych niewystarczającymi w poważniejszych zadaniach; plikowa baza danych nie jest odpowiednia nawet dla niewielkiej firmy, a co dopiero potężnej korporacji. Co innego relacyjne bazy danych, określane często skrótem RSZBD (relacyjne systemy zarządzania bazami danych). Te pozwalają na określenie powiązań pomiędzy poszczególnymi elementami danych; dane te przechowywane są w tabelach a w ramach tabel w polach, podobnie trochę jak to ma miejsce w przypadku arkusza kalkulacyjnego. Dane te można efektywnie przeszukiwać i porządkować. Z tego względu przedmiotem zainteresowania niniejszego rozdziału będą właśnie relacyjne bazy danych.

Przykładami relacyjnych baz danych są Oracle, DB2, Microsoft SQL Server czy rozprowadzane nieodpłatnie PostgreSQL i MySQL. Omówienie obejmować będzie dwie ostatnie — Czytelnik będzie miał okazję poznać podstawy administrowania bazami danych, pozna też podstawy języka SQL, będącego standardową metodą manipulowania danymi przechowywanymi w bazach danych.

Zasada działania relacyjnych baz danych

Prawidłowe utworzenie i efektywne zarządzanie relacyjną bazą danych wymaga od administratora bazy znajomości podstawowych zasad regulujących sposób przechowywania informacji w takich bazach danych; przyda się też wiedza na temat tego, jak użytkownicy odwołują się do serwera bazy danych i za jego pośrednictwem manipulują danymi. Punkt ten nie zawiera co prawda szczegółowego wykładu teoretycznego na temat zasad funkcjonowania baz danych, ale zawarte tu informacje powinny wystarczyć do przyswojenia sobie choćby podstawowych koncepcji.

Relacyjny system zarządzania bazą danych przechowuje dane w tabelach, które można sobie wyobrazić jako arkusze kalkulacyjne. Każda z kolumn tabeli to tzw. pole tabeli; kolumna może np. zawierać nazwisko czy adres. Każdy wiersz tabeli to pojedynczy rekord tabeli (zwany też krotką). Tabela posiada nazwę, za pośrednictwem której można się do niej odwoływać, pozyskując z niej dane bądź umieszczając w niej nowe informacje. Przykładową prostą tabelę relacyjnego systemu zarządzania bazą danych ilustruje rysunek 17.1.

Imię	Nazwisko	Adres	Miasto	Stan	Kod	Telefon
John	Doe	501 Somestreet	Anytown	NY	55011	555-555-1212
Jane	Doe	501 Somestreet	Anytown	NY	55011	555-555-1212
John	Palmer	205 Anystreet	Sometown	NY	55055	123-456-7890
Robert	Johnson	100 Easystreet	Easytown	CT	12345	111-222-3333

Rysunek 17.1. *Powyższa ilustracja pokazuje sposób przechowywania danych — prezentowana tabela zawiera cztery wiersze (rekordy), z których każdy zawiera po siedem pól (kolumn)*

W przykładzie prezentowanym na rysunku 17.1 baza danych zawiera tylko jedną tabelę. Większość systemów relacyjnych baz danych utrzymuje jednak dużo bardziej złożone struktury. Na rysunku 17.2 widać przykład bazy danych o nazwie `baza_danych`, zawierającej dwie tabele.

baza_danych

Telefony

Nazwisko	Imię	Telefon
Doe	John	555-555-1212
Doe	Jane	555-555-1212
Palmer	John	123-456-7890
Johnson	Richard	111-222-3333

Ssaki_serengeti

Nazwa potoczna	Rząd	Rodzina	Rodzaj	Gatunek	Podgatunek
Lew afrykański	drapieżne	kotowate	Panthera	leo	
Słoń afrykański	trąbowce	słoniowate	Loxodonta	africana	
Surykatka	drapieżne	łaszowate	Suricata	suricatta	
Hiena	drapieżne	hienowate	Crocuta	croucuta	
Żyrafa Thomsona	parzystokopytne	żyrafy	Giraffa	cameloparalis	retciculata
Zebra stepowa	parzystokopytne	koniowate	Equus	burchelli	bohmi
Gepard	drapieżne	kotowate	Acinonyx	jubatus	
Gazela Thomsona	parzystokopytne	krętorogie	Gazella	thomsoni	
Hipopotam	parzystokopytne	hipopotamowate	Hippopotamus	amphibius	

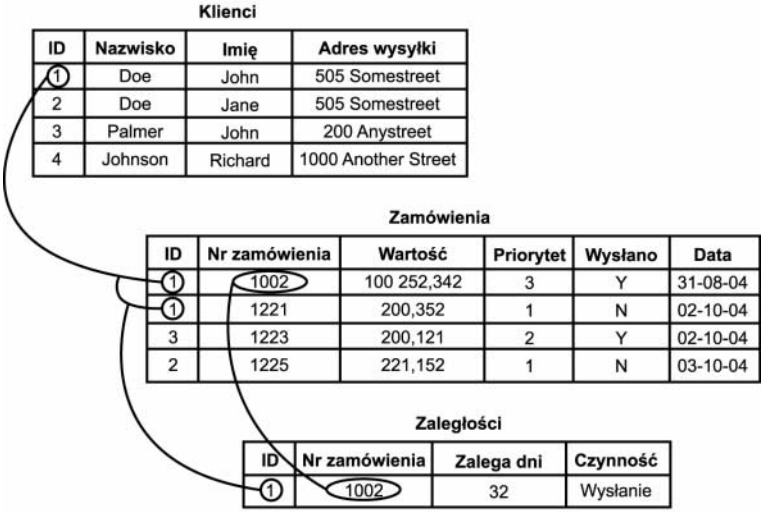
Rysunek 17.2. *Pojedyncza baza danych może zawierać dwie tabele; tutaj telefony oraz ssaki_serengeti*

W przykładowej bazie danych `baza_danych` tabela `telefony` zawiera cztery rekordy (wiersze), z których każdy podzielony jest na trzy pola (kolumny) danych. Tabela `ssaki_serengeti` zawiera z kolei dziewięć rekordów, podzielonych na sześć pól.

Jeżeli Czytelnik podejrzewa, że obie te tabele (a konkretnie zawarte w nich dane) nie mają ze sobą zupełnie nic wspólnego, to ma rację. W relacyjnym systemie baz danych użytkownik może tworzyć w jednej bazie danych wiele tabel, również wtedy, kiedy tabele te nie są nijak ze sobą powiązane.

W ramach przykładu poprawnego wykorzystania możliwości relacyjnej bazy danych wyobraźmy sobie niewielką firmę sprzedającą jakiś wyimaginowany produkt i dysponującą skomputeryzowaną bazą danych klientów. W bazie tej obok nazwisk, adresów i numerów telefonów klientów powinny znajdować się informacje o składanych przez nich zamówieniach i wystawionych fakturach. Model taki można zaimplementować za pomocą trzech tabel relacyjnej bazy danych. Przykład struktury takiej bazy danych prezentowany na rysunku 17.3.

W przykładzie z rysunku 17.3 każdy rekord konsumenta zawiera pole z indywidualnym identyfikatorem klienta. Pole to przechowuje niepowtarzalny numer, na podstawie którego można danego klienta skojarzyć z innymi informacjami, co pozwoli na określenie zamówień i stanu płatności danego klienta. Niepowtarzalność identyfikatora klienta jest bardzo ważną cechą — dwaj klienci mogą się tak samo nazywać, mieszkać w jednym domu i korzystać z tego samego telefonu, ale muszą być oznaczeni różnymi numerami



Rysunek 17.3. *Za pomocą trzech powiązanych wzajemnie tabel można stworzyć system bazy danych śledzących zamówienia i płatności wszystkich klientów*

identyfikującymi. W tabelach `Zamówienia` i `Zaległości` zastosowanie identyfikatora klienta pozwala na uniknięcie dublowania informacji (nazwiska, adresu itd.). W prezentowanym układzie wyszukiwanie zamówień czy zaległości w płatnościach dowolnego z klientów bazuje na pojedynczym kluczu wyszukiwania (identyfikatorze klienta), zamiast na wielu takich kluczach (nazwisko, adres itd.). Rezultaty są więc zwracane dużo szybciej, a samo wyszukiwanie łatwiejsze w implementacji.

Mając pewne wyobrażenie o sposobie działania relacyjnych baz danych można przystąpić do nauki sposobu wprowadzania i pozyskiwania danych z baz danych. Wymaga to bowiem znajomości języka SQL.

Podstawy języka SQL

SQL to język zapytań do serwerów baz danych, „rozumiany" przez chyba wszystkie dostępne na rynku serwery relacyjnych baz danych. Zapytania języka SQL służą do konstruowania kryteriów pobierania danych i definiowania danych wprowadzanych do bazy. Przy czym tak, jak we wszystkich językach programowania, zapytania języka SQL muszą odpowiadać pewnej składni, definiującej strukturę i znaczenie poszczególnych elementów zapytania.

W większości przypadków język SQL służy do konstruowania zapytań osadzanych w kodzie aplikacji klienckich, tak, aby użytkownicy nie musieli sami konstruować zapytań. Z tego względu większość użytkowników baz danych nie jest świadoma pośrednictwa języka SQL i jego roli — zresztą słusznie; użytkownicy końcowi nie muszą przecież nic wiedzieć o strukturze zapytań SQL.

Jednak administrator bazy danych powinien znać przynajmniej podstawy języka SQL, nawet, jeśli w zakresie jego obowiązków nie leżą zadania typowo programistyczne. Na szczęście język SQL bardzo przypomina język naturalny (oczywiście angielski — *przyp. tłum.*), co znakomicie ułatwia jego opanowanie, przynajmniej w podstawowym stopniu.

Kolejne akapity w żadnym razie nie aspirują do miana podręcznika języka SQL. Mają po prostu wprowadzić Czytelnika w absolutne podstawy tego języka. Czytelnicy zainteresowani bardziej szczegółowymi informacjami o programowaniu w języku SQL powinni zajrzeć do jednej z wielu książek szerzej traktujących to zagadnienie, z publikacją *MySQL* (New Riders Publishing,) czy *Sams Teach Yourself MySQL in 21 Days* albo *PostgreSQL* (Sams Publishing)[1].

Tworzenie tabel

Relacyjny system zarządzania bazą danych przechowuje dane w tabelach, przypominających nieco te, znane z arkuszy kalkulacyjnych. Zanim jednak będzie można umieścić dane w tabeli, należy taką tabelę utworzyć, definiując jej kolumny. Służy do tego zapytanie `CREATE`.

Dla przykładu, tabela `ssaki_serengeti` z rysunku 17.2, posiada sześć kolumn (pól): `nazwa_potoczna`, `rząd`, `rodzina`, `rodzaj`, `gatunek` oraz `podgatunek`.

Język SQL pozwala na przypisywanie do każdej z kolumn tabeli konkretnego typu danych, określającego charakter informacji przechowywanych w tej kolumnie. Wśród dostępnych typów są `INT` (liczba całkowita), `FLOAT` (liczba zmiennoprzecinkowa), `CHAR` (ciąg znaków o ustalonej długości) oraz `VARCHAR` (ciąg znaków o zmiennej długości).

Dostępnych jest też kilka typów specjalnych, jak `DATE` (data) czy `ENUM` (zbiór wyliczeniowy). Pole typu `ENUM` przyjmuje wartości wyłącznie ze zdefiniowanego wcześniej zbioru. Można taki zbiór wykorzystać np. do określania płci zwierzęcia wpisywanego do bazy danych — kod płci może zostać oparty na zbiorze wyliczeniowym, zawierającym tylko dwa elementy: `F` (od łac. *femininum*, gram. rodzaj żeński) oraz `M` (od łac. *masculinum*, gram. rodzaj męski).

Już pobieżnie analizując tabelę `ssaki_serengeti` można stwierdzić, że kolumny tej tabeli zawierają dane w postaci wyłącznie ciągów znakowych. Ciągi te cechują się różną liczbą znaków. Na tej podstawie można zdecydować o przypisaniu do tych kolumn typu `VARCHAR`. Tabela `ssaki_serengeti` ma jeszcze jedną ciekawą właściwość. Otóż wszystkie kolumny każdego wiersza, poza kolumną ostatnią, muszą zawierać jakieś wartości. Ostatnia kolumna może pozostać pusta, ponieważ nie wszystkie zwierzęta wymienione w tabeli mają przypisaną klasyfikację podgatunku. Oba te spostrzeżenia przydadzą się przy tworzeniu tabeli.

Jak już wspomniano, do tworzenia tabeli służy zapytanie `CREATE` języka SQL. Zapytanie to ma następującą składnię:

```
CREATE TABLE nazwa_tabeli (nazwa_kolumny, typ_kolumny(parametry_typu), opcje, …);
```

Przed przystąpieniem do konstruowania zapytania `CREATE` warto wspomnieć o dwóch rzeczach:

[1] Albo np. *Relacyjne bazy danych* Marka Whitehorna i Billa Marklyna (Helion, 2003), czy *SQL. Księga eksperta* Hansa Ladanyi'a (Helion, 2000) — *przyp. tłum.*

- Zapytania języka SQL mogą zawierać zarówno wielkie, jak i małe litery. Prawidłowe jest więc zarówno zapytanie pisane w całości literami wielkimi (`CREATE TABLE ...`), jak i literami małymi (`create table ...`) bądź o mieszanej wielkości (`Create Table ...`). Warto jednak wyrobić sobie nawyk pisania słów kluczowych języka SQL literami wielkimi; przyda się to podczas pisania odwołujących się do bazy danych aplikacji w języku C czy Perl — znakomicie ułatwia to późniejszą analizę kodu źródłowego (słowa kluczowe w popularnych językach programowania pisane są niemal zawsze małą literą).
- Nie trzeba się przejmować liczbą odstępów pomiędzy kolejnymi elementami zapytania. Będzie się można o tym przekonać przy okazji tworzenia tabeli `ssaki_serengeti`.

Poniższe zapytanie SQL tworzy tabelę `ssaki_serengeti` według schematu z rysunku 17.2:

```
CREATE TABLE ssaki_serengeti
(
 nazwa_potoczna VARCHAR(25) NOT NULL,
 rzad VARCHAR(25) NOT NULL,
 rodzina VARCHAR(25) NOT NULL,
 rodzaj VARCHAR(25) NOT NULL,
 gatunek VARCHAR(25) NOT NULL,
 podgatunek VARCHAR(25) NULL,
);
```

Jak widać, zapytanie kończy się znakiem średnika. Znak ten stanowi sygnał końca każdego zapytania języka SQL (niekiedy można go jednak pominąć — przykłady takich sytuacji zostaną zaprezentowane w dalszej części rozdziału).

Wypełnianie tabel danymi

Po utworzeniu tabel można rozpocząć ich wypełnianie informacjami. Można to robić ręcznie, konstruując kolejne zapytania `INSERT`, zgodnie z następującą składnią:

```
INSERT INTO nazwa_tabeli VALUES ('wartość1', 'wartość2', 'wartość3', …);
```

Wykonanie zapytania `INSERT` powoduje wstawienie do kolejnych kolumn tabeli *`nazwa_tabeli`* wartości *`wartość1`*, *`wartość2`*, *`wartość3`* i tak dalej. Wartości te tworzą pojedynczy wiersz (rekord) tabeli. Podawane wartości trafiają do kolejnych kolumn wiersza, chyba że w zapytaniu określone zostaną wprost kolumny docelowe (np. wtedy, kiedy wiersz określony jest tylko dla niektórych kolumn). Kolumny docelowe dla wartości określa się następująco:

```
INSERT INTO nazwa_tabeli (kolumna1, kolumna2) VALUES ('wartość1', 'wartość2');
```

Pojedynczym zapytaniem `INSERT` można też wypełnić więcej niż jeden wiersz; składnia takiego zapytania wygląda następująco:

```
INSERT INTO nazwa_tabeli VALUES ('wartość1', 'wartość2'), ('wartość3', 'wartość4');
```

W powyższym zapytaniu wartości *`wartość1`* oraz *`wartość2`* tworzą wiersz pierwszy, wartości *`wartość3`* i *`wartość4`* należą do drugiego wstawianego wiersza.

Poniższy przykład ilustruje sposób wprowadzenia do tabeli wiersza opisującego lwa afrykańskiego:

```
INSERT INTO ssaki_serengeti VALUES('Lew afrykański', 'Drapieżne', 'Kotowate',
↳'Panthera', 'Leo', NULL);
```

W bazie danych MySQL, jeśli wiersz nie ma definiować któregoś z ostatnich pól, należy w miejscu wartości dla tego pola podać wartość pustą (`NULL`). W przypadku bazy danych PostgreSQL ostatnie kolumny mogą po prostu zostać pominięte. Rzecz jasna, kolumny środkowe muszą być zawsze określone wprost — jeśli nie mają wartości, należy zamiast niej podać wartość `NULL`.

Zwykle zapytanie `INSERT` osadzone jest w aplikacji stanowiącej interfejs bazy danych, aby użytkownicy wprowadzający dane do bazy nie musieli samodzielnie konstruować zapytań.

Pobieranie informacji z bazy danych

Po to tworzy się bazy danych, aby można było efektywnie i wygodnie korzystać z przechowywanych w nich informacji, pobierając je, porządkując i tworząc na ich podstawie rozmaite zestawienia. Podstawowa operacja pobierania danych implementowana jest w języku SQL zapytaniem `SELECT`, o następującej składni:

```
SELECT kolumna(-ny) FROM nazwa_tabeli WHERE kryterium_wyszukiwania;
```

Pierwsze dwa elementy zapytania — klauzula `SELECT` oraz klauzula `FROM` — są elementami wymaganymi. Klauzula `WHERE` jest elementem opcjonalnym. W przypadku jego braku zapytanie wybiera wszystkie wiersze tabeli *`nazwa_tabeli`*.

Kolumny to wyliczenie nazw kolumn, które mają zostać uwzględnione w zbiorze danych wyjściowych. W przypadku, kiedy do zbioru wynikowego ma trafić więcej niż jedna kolumna, nazwy kolumn należy w klauzuli `SELECT` rozdzielić przecinkami. Zamiast wypisywać nazwy wszystkich kolumn po kolei, można zastosować symbol wieloznaczny — gwiazdkę (*). Dla przykładu, poniższe zapytanie spowoduje wyświetlenie wszystkich kolumn tych wierszy tabeli, które spełniają kryteria wyszukiwania:

```
SELECT * FROM ssaki_serengeti;
```

Jeśli natomiast wydruk ma zawierać np. tylko nazwy potoczne gatunków, zapytanie powinno przyjąć postać:

```
SELECT nazwa_potoczna FROM ssaki_serengeti;
```

Zbiór wynikowy może też zawierać np. tylko rodzaj i nazwę gatunku:

```
SELECT rodzaj, gatunek FROM ssaki_serengeti;
```

Tyle, że wynik powyższego zapytania raczej nie zadowoli biologów. Otóż nazwa zwierzęcia składa się zawsze z nazwy rodzaju i gatunku. Dopiero one tworzą pełną nazwę. Na szczęście zarówno MySQL, jak i PostgreSQL oferują funkcję łączenia ciągów. W obu bazach danych funkcja ta jest implementowana nieco inaczej.

W bazie danych MySQL do połączenia wartości kolumn `rodzaj` i `gatunek` można wykorzystać funkcję `CONCAT`, jak poniżej:

```
SELECT CONCAT(rodzaj, " ", gatunek) AS Pełna_Nazwa FROM ssaki_serengeti;
```

Powyższe zapytanie spowoduje wyświetlenie połączonych wartości kolumn w kolumnie o nazwie `Pełna_Nazwa`. W wywołaniu funkcji `CONCAT` drugim argumentem jest ciąg zawierający pojedynczy znak spacji. Gdyby go tam zabrakło, wartości kolumn zostałyby ze sobą „sklejone", co nie wyglądałoby najlepiej na wydruku.

W bazie danych PostgreSQL funkcja łączenia ciągów wywoływana jest dwoma symbolami potoku (`||`). Odpowiednikiem prezentowanego powyżej zapytania w wersji dla bazy PostrgeSQL byłoby więc następujące zapytanie:

```
SELECT (rodzaj||' '||gatunek) AS Pełna_Nazwa FROM ssaki_serengeti;
```

Tutaj ujmowanie łączonych podciągów w nawiasy jest nieobowiązkowe, choć zwiększa czytelność zapytania. Jak poprzednio, również tym razem ujęty w znaki (tutaj pojedynczego) cudzysłowu znak odstępu pomiędzy łączonymi ciągami wprowadza spację rozdzielającą podciągi optycznie. Brak tego znaku spowodowałby rzecz jasna sklejenie obu podciągów do postaci pojedynczego wyrazu.

Zazwyczaj przy pobieraniu danych interesuje nas jedynie podzbiór wszystkich danych przechowywanych w tabeli czy bazie danych. Zwykle taki podzbiór konstruuje się z wierszy o pewnych charakterystycznych cechach. Wyszukiwanie według tych cech można wymusić, umieszczając w zapytaniu `SELECT` klauzulę `WHERE`. Dla przykładu, aby wyszukać w tabeli `ssaki_serengeti` wszystkie ssaki drapieżne, należałoby skonstruować zapytanie następującej treści:

```
SELECT * FROM ssaki_serengeti WHERE rzad="Drapieżne";
```

Jeśli tabela `ssaki_serengeti` zawiera dane takie, jak na rysunku 17.2, wynikiem zapytania powinny być wiersze opisujące lwa afrykańskiego, surykatkę, hienę i geparda. To bardzo proste zapytanie — język SQL pozwala na definiowanie dużo bardziej złożonych operacji wyszukiwania. Zapytania takie mogą zawierać w klauzuli `WHERE` warunki logiczne, określane operatorami `AND` (logiczny iloczyn) oraz `OR` (logiczna suma). Jeśli na przykład z uzyskanego poprzednio zbioru wynikowego chcielibyśmy wyodrębnić wyłącznie te drapieżne, które równocześnie nie należą do rodziny hienowatych, zapytanie powinno przybrać następującą postać:

```
SELECT * FROM ssaki_serengeti WHERE rzad="Drapieżne" AND rodzina!="Hienowate";
```

W programowaniu operator `!=` oznacza zazwyczaj „różny od". Jeśli za tabelę źródłową przyjąć tabelę z rysunku 17.2, to zbiór wynikowy zapytania zawierać będzie wiersze lwa afrykańskiego, surykatki i geparda — hiena, jako hienowata, nie trafi do zbioru wynikowego.

W przypadku, gdyby zbiór wynikowy miał obejmować wszystkie drapieżne i parzystokopytne z wyjątkiem tych, które należą do rodzin hienowatych oraz hipopotamowatych, należałoby zastosować następującą kombinację operatorów `AND` i `OR`:

```
SELECT * FROM ssaki_serengeti WHERE rzad="Drapieżne" OR rzad="Parzystokopytne"
↳AND rodzina!="Hienowate" AND rodzina!="Hipopotamowate";
```

Powyższe zapytanie wybierze do zbioru wynikowego lwa afrykańskiego, surykatkę, geparda, żyrafę Thomsona, zebrę stepową, geparda i gazelę Thomsona. Pominięte zostaną natomiast hipopotam oraz hiena i słoń afrykański.

Jednym z najczęstszych błędów programistów aplikacji obsługujących bazy danych jest niewłaściwe zastosowanie operatorów logicznych. Na przykład, jeśli ktoś wyrazi życzenie: „Znajdź wszystkie zwierzęta rodzaju Panthera oraz Acinonyx", to z pozoru żądanie to przełożone wprost na język SQL powinno spowodować wybranie z tabeli wierszy opisujących lwa afrykańskiego i geparda. Jakież będzie zaskoczenie programisty, gdy zapytanie to nie zwróci żadnego wiersza! A stanie się tak, ponieważ po przełożeniu żądania wprost na język SQL kryterium wyszukiwania okazuje się przynależność gatunku do dwóch rodzajów, co w systematyce gatunków jest, rzecz jasna, nie do pomyślenia. Niezależnie więc od liczby wierszy, takie zapytanie nie zwróci żadnego z nich. W miejsce oraz (`AND`) należy więc zastosować operator lub (`OR`).

Oczywiście powyższa prezentacja nie wyczerpuje wszystkich możliwości języka SQL. Ale, jak już wspomniano, podrozdział ten nie ma stanowić podręcznika języka SQL; jego zadaniem było wyłącznie zaznajomienie Czytelnika z absolutnymi podstawami tego języka, niezbędnymi każdemu szanującemu się administratorowi baz danych.

Wybór bazy danych: MySQL kontra PostgreSQL

Pierwszym krokiem do korzystania z baz danych w Linuksie jest wybór serwera bazy danych, odpowiadającego konkretnym potrzebom. Dla systemu Linux powstało wiele implementacji serwerów baz danych; są wśród nich zarówno programy dostępne nieodpłatnie, jak i te kosztujące tysiące dolarów. Te ostatnie, jak baza danych Oracle, nie będą jednak omawiane w niniejszej książce. Ten podrozdział zostanie w całości poświęcony dwóm darmowym systemom zarządzania bazami danych: MySQL oraz PostgreSQL.

W obu przypadkach mamy do czynienia z produktami wysokiej jakości; oba umieszczone są na dołączonych do książki płytach CD-ROM. Oba też, pomimo tego, że są rozprowadzane nieodpłatnie, dają sobie radę nawet z bardzo złożonymi projektami. Wykorzystywane są powszechnie w organizacjach komercyjnych, agencjach rządowych (z MySQL korzysta na przykład NASA), instytutach badawczych i placówkach edukacyjnych.

Obie bazy danych są bardzo zaawansowane i prawdopodobnie każda z nich będzie spełniać potrzeby Czytelnika. Różnią się jednak od siebie na tyle, że wybór jednej z nich może wpłynąć na efektywność dostępu do danych bądź prostotę programowania aplikacji bazodanowych. Niektóre z podstawowych cech obu baz danych, jak i różnice w implementacji poszczególnych funkcji, zostaną omówione w kolejnych punktach; informacje te powinny pomóc w wyborze jednej z nich dla konkretnego projektu.

Szybkość

Do niedawna kryterium to wskazywało jednoznacznie na bazę danych MySQL. Baza ta cieszy się opinią bardzo szybkiej. Do niedawna też baza PostgreSQL wypadała w tym porównaniu blado.

Nowsze wersje bazy danych PostgreSQL zostały jednak znacznie ulepszone (przede wszystkim w zakresie wydajności operacji dostępu do dysku i porządkowania danych). W pewnych sytuacjach, na przykład w okresach największego natężenia żądań dostępu do danych, baza danych oparta na PostgreSQL okazuje się wręcz szybsza od MySQL, co będzie można sprawdzić w następnym podrozdziale. Mimo wszystko sława MySQL wciąż jest jak najbardziej zasłużona.

Blokowanie danych

W bazie danych stosuje się zapobiegający naruszeniu integralności danych mechanizm, polegający na blokowaniu danych na czas dostępu do nich. Póki dane są zablokowane przez jeden proces, inny proces nie może się do nich odwołać. Wszystkie procesy próbujące odwoływać się do zablokowanych danych muszą poczekać na zwolnienie blokady. Po jej zwolnieniu blokadę zakłada następny proces, a reszta dalej musi czekać na swoją kolej.

Oczywiście operacje na bazie danych zajmują zwykle bardzo niewiele czasu, więc w środowiskach o niewielkiej liczbie współbieżnych procesów odwołujących się do bazy danych blokady są zakładane na stosunkowo krótkie okresy, nie wprowadzające żadnych zauważalnych opóźnień dostępu. Jednak już w środowisku, w którym ze wspólnej bazy danych korzysta wielu użytkowników, blokowanie może doprowadzić do zmniejszenia wydajności ich pracy.

W bazie danych MySQL blokowanie implementowane jest w sposób zasadniczo odmienny od przyjętego w bazie danych PostgreSQL.

W starszych wersjach MySQL blokady obejmowały całe tabele, co w przypadku dużego natężenia odwołań prowadziło do powstawania zatorów. Kiedy bowiem jeden z użytkowników modyfikował pojedynczy wiersz tabeli, nikt inny nie mógł równocześnie wprowadzać innych wierszy. Jeśli tabela zawierała, powiedzmy 500 000 wierszy, to na czas realizacji operacji na dowolnym z tych wierszy pozostałe pozostawały niedostępne. W środowiskach z niewielką liczbą użytkowników nie stanowiło to większego problemu, ponieważ większość operacji modyfikacji wiersza przebiega w bardzo krótkim czasie. Jednak przy większej liczbie użytkowników blokowanie całych tabel mogło doprowadzić do zauważalnych opóźnień.

W przypadku bazy danych PostgreSQL blokowanie obejmowało zawsze wyłącznie pojedyncze wiersze. Blokowany jest zawsze ten wiersz, który jest aktualnie modyfikowany. Reszta tabeli może w tym czasie być dostępna dla pozostałych użytkowników. Blokowanie z dokładnością do pojedynczych wierszy znacząco redukuje opóźnienia wynikające ze współbieżnego dostępu do tabeli w środowiskach wielodostępnych. Stąd wniosek, że w środowiskach cechujących się znacznym natężeniem żądań dostępu do danych PostgreSQL może sprawdzać się lepiej niż MySQL.

Implementacja mechanizmu blokowania w bazie danych MySQL omówiona jest szczegółowo na stronie *http://www.mysql.com/doc/en/Internal_locking.html*. Dystrybucja Fedora zawiera pakiet MySQL w wersji 3.23.58-9, ale już wersja 4.0 (dostępna w czasie przygotowywania wydania) zawiera mechanizm blokowania z dokładnością do pojedynczych wierszy. Dostępna jest również do pobrania wersja beta 4.1 oraz wersja alfa 5.0 pakietu MySQL.

Mechanizm blokowania stosowany w bazie danych PostgreSQL omówiony jest w dokumencie dostępnym pod adresem *http://www.postgresql.org/docs/7.3/interactive/mvcc.html*.

Przetwarzanie transakcji a ochrona spójności danych — reguły ACID

Bazy danych MySQL i PostgreSQL różnią się również jakością ochrony spójności danych. Skrót ACID rozszyfrowywany jest do postaci czterech reguł, które taką spójność zapewniają:

- **Niepodzielność** (ang. *atomicity*) — zwana też atomowością, odnosi się do głównej cechy atomów, jaką miała być właśnie ich niepodzielność. W kontekście baz danych oznacza to możliwość traktowania szeregu operacji jako niepodzielnych, zwanych też **transakcjami**. Transakcja może zostać wykonana albo w całości, albo wcale. Jeżeli którakolwiek z operacji składowych transakcji nie zostanie wykonana, należy unieważnić całą transakcję.
- Jeśli, na przykład, w ramach transakcji dojdzie do usunięcia pierwotnego wiersza, ale nie uda się — czy to w wyniku awarii zasilania, czy też załamania serwera — zrealizować operacji wprowadzenia nowego wiersza, niepodzielność transakcji powinna zagwarantować przywrócenie wartości pierwotnej wierszowi, którego aktualizacja została zakłócona.
- **Spójność** (ang. *consistency*) — oznacza, że żadna z transakcji nie może spowodować przejścia bazy danych ze stanu spójności do stanu niespójności. Stan niespójności objawia się naruszeniem zdefiniowanych dla danych relacji i może być spowodowany błędem w aplikacji odwołującej się do danych, przekłamaniami transmisji sieciowych i tym podobnymi. Gwarancja spójności to gwarancja, że każda z transakcji, która miałaby naruszyć spójność bazy danych, zostanie wycofana. Zapobiega to na przykład usunięciu wiersza czy pola, do którego odnoszą się inne wiersze bazy danych (usunięcie takie spowodowałoby „osierocenie" owych wierszy).
- **Izolacja** (ang. *isolation*) — izolacja to gwarancja pełnej niezależności wszystkich transakcji. W przypadku dwóch transakcji manipulujących tymi samymi danymi nie może dojść do sytuacji, w której modyfikacje wprowadzane przez jedną transakcję zakłócają działanie drugiej. Sposób implementacji mechanizmu izolacji jest zwykle konfigurowany przez programistę bazy danych. Jednym z podstawowych środków zapewniania izolacji jest blokowanie danych.
- **Trwałość** (ang. *durability*) — po zatwierdzeniu transakcji wprowadzone przez nią modyfikacje danych nie mogą zostać utracone z powodu awarii zasilania czy błędu systemu, sieci i tak dalej. Trwałość jest zwykle gwarantowana mechanizmem

dzienników transakcji. W przypadku awarii komputera serwer bazy danych, po załączeniu zasilania, może przeanalizować zawartość plików dziennika transakcji i utrwalić te wprowadzone przez transakcje zmiany, których nie udało się zapisać na dysk przed awarią.

Baza danych PostgreSQL spełnia wszystkie cztery kryteria ACID; MySQL szybko zaś nadrabia zaległości. Dla przykładu, najnowsza wersja (dostępna w ramach projektu Fedora Project oraz na stronie domowej PostgreSQL, dostępnej pod adresem *http://www.mysql.com/*) obsługuje już transakcje, co oznacza mniejsze ryzyko utraty danych w wyniku awarii. Wielu programistów uważa jednak, że to właśnie PostgreSQL jest pod tym względem lepiej przystosowany do środowisk z wieloma użytkownikami i wielką liczbą regularnie aktualizowanych danych.

Firma rozprowadzająca MySQL Server, czyli MySQL AB zaleca regularne wykonywanie kopii zapasowych bazy danych MySQL. W nowszych wersjach bazy zaimplementowane zostały jednak mechanizmy zabezpieczające — wersja 4.0 zawiera szereg udoskonaleń implementacji reguł ACID, w tym automatyczne przywracanie danych po awarii. Więcej informacji na ten temat zawiera dokumentacja publikowana w witrynie *http://www.mysql.com/*.

Podzapytania SQL

Zastosowanie podzapytań pozwala na połączenie w jedną kilku operacji, przy czym operacje te mogą odwoływać się do zbiorów wynikowych pozostałych operacji. Podzapytania umożliwiają konstruowanie naprawdę złożonych zapytań SQL. Ich zastosowanie eliminuje też ryzyko modyfikacji danych, które mogłyby zajść pomiędzy poszczególnymi operacjami, gdyby były one wykonywane osobno. Podzapytania obsługiwane są tylko przez bazę danych PostgreSQL; w bazie MySQL możliwość ta jest niedostępna, choć istnieje kilka sposobów symulacji podzapytań.

Import danych tabeli z plików tekstowych

W obu omawianych bazach danych zapytania tworzące tabelę bazy danych można wprowadzać z poziomu wiersza polecenia programów-klientów bazy danych. Operacja taka jest jednak dość ryzykowna: w przypadku najmniejszej nawet pomyłki w tekście zapytania trzeba całość pisać od nowa. Na szczęście zarówno MySQL, jak i PostgreSQL, potrafią też odczytywać zapytania SQL ze zwykłych plików tekstowych. Można więc co dłuższe i częściej wykorzystywane zapytania zapisać w zwykłym pliku tekstowym, nadając mu ewentualnie rozszerzenie *.sql*, informujące o zawartości pliku.

Zapytania SQL zapisane w pliku tekstowym (tu *ssaki.sql*) można wprowadzać do bazy danych za pośrednictwem następujących poleceń (pierwsze z nich dotyczy bazy danych MySQL, drugie — bazy danych PostgreSQL):

```
$ mysql nazwa_bazy_danych < ssaki.sql
$ psql nazwa_bazy_danych < ssaki.sql
```

Oczywiście, aby takie polecenia miały szansę zadziałać, muszą być wykonywane w imieniu użytkownika dysponującego odpowiednimi uprawnieniami w bazie danych *nazwa_bazy_danych*. Patrz podrozdział „Przyznawanie i odbieranie uprawnień w bazie danych MySQL” oraz „Tworzenie kont użytkowników bazy danych MySQL”

Języki proceduralne i wyzwalacze

Język proceduralny to w tym kontekście zewnętrzny język programowania, który można wykorzystać do programowania funkcji i procedur bazy danych. Można w ten sposób implementować funkcje niedostępne z poziomu języka SQL. **Wyzwalacz** z kolei to procedura lub funkcja zewnętrzna, która jest uruchamiana w reakcji na określone zdarzenie bazy danych. Wyzwalacz może na przykład zgłaszać wyjątek, kiedy zapytanie `INSERT` definiuje dla którejś z kolumn nieoczekiwaną bądź niepoprawną wartość.

Dla przykładu, w bazie danych wykorzystywanej do śledzenia populacji zwierząt na pewnym obszarze, można wykorzystać wyzwalacz do podnoszenia wyjątku w sytuacji, kiedy użytkownik wprowadzi współrzędne obserwacji, które nie mają (z punktu widzenia obszaru obserwacji) żadnego sensu. W bazie danych PostgreSQL można taki wyzwalacz napisać w języku proceduralnym o nazwie PL/pgSQL. MySQL dysponuje co prawda ograniczoną liczbą procedur wbudowanych i wyzwalaczy, jednak nie udostępnia programiście własnego języka proceduralnego. Oznacza to, że w bazach danych MySQL nie można stosować wyzwalaczy; podobne efekty można jednak osiągnąć przez pomysłowe programowanie aplikacji klienckich.

Dostępne aplikacje

W tym momencie Czytelnik może stwierdzić, że dalsze porównywanie, z uwagi na wykryte dotychczas braki MySQL, nie ma większego sensu. Nie jest to jednak prawdą, ponieważ MySQL ma zaletę, której brak bazie PostgreSQL — dużą liczbę gotowych aplikacji. Dla przykładu, aby utworzyć na stronie WWW forum dyskusyjne, można skorzystać z wykorzystującego bazę danych MySQL programu Phorum. Pozwala to na zaoszczędzenie mnóstwa pracy, która byłaby potrzebna w przypadku samodzielnego implementowania forum.

Choć faktycznie większość z opisywanych wcześniej funkcji jest w bazie MySQL niedostępna, to na rynku dostępna jest ulepszona wersja bazy danych, o nazwie MySQL-Max, przystosowana do obsługi środowisk produkcyjnych opartych na systemie Linux z najnowszą wersją jądra. Więcej informacji o tym produkcie publikowanych jest w witrynie *http://www.mysql.com/*.

Bazy danych i Fedora — pierwsze kroki

Aby udostępnić użytkownikom systemu Fedora usługi baz danych, należy wykonać szereg czynności, z których pierwszą jest poprawne uruchomienie serwera bazy danych (zwykle realizowane w ramach rozruchu systemu). Użytkownik *root* musi więc upewnić się, czy serwer bazy danych jest uwzględniony na liście usług do uruchomienia na poszczególnych poziomach uruchomieniowych (patrz rozdział 7., „Zarządzanie usługami systemowymi"). Następnie należy utworzyć konto użytkownika głównego bazy danych i zainicjalizować serwer (w przypadku baz danych MySQL polega to na wykonaniu polecenia `mysql_install_db`; szczegóły opisane są w punkcie „Inicjalizowanie katalogu danych bazy MySQL"). Użytkownik główny bazy danych musi otrzymać hasło, po czym należy utworzyć jedną lub kilka baz danych. Dopiero wtedy użytkownik główny bazy danych może przystąpić do wprowadzania użytkowników i przypisywania im uprawnień w ramach utworzonych baz danych.

Instalowanie i konfigurowanie bazy danych MySQL

Dystrybucja Fedora zawiera stabilną, darmową wersję bazy danych MySQL. Oprogramowanie bazy MySQL można też pobrać z witryny *http://www.mysql.com/*. Oprogramowanie to jest tam udostępniane w postaci kodu źródłowego oraz wersji skompilowanej, jak również w postaci pakietu RPM. Instalację oprogramowania MySQL można przeprowadzić w ramach pierwotnej instalacji systemu, albo później skorzystać z programu `system-config-packages`. Sposób instalowania pakietów RPM pokazany był w rozdziale 8.

W tym podrozdziale opisana zostanie jedynie procedura instalacji oprogramowania MySQL z kodu źródłowego i postaci skompilowanej; opis dotyczyć będzie wersji 4.0.

Dużo prostszą procedurą jest instalacja oprogramowania MySQL z pakietu RPM. W takim przypadku wszystkie niezbędne skrypty uruchomieniowe są instalowane automatycznie; automatycznie są też tworzone niezbędne do działania bazy konta użytkowników oraz grupy użytkowników. Instalacja z kodu źródłowego czy postaci skompilowanej jest znacznie bardziej pracochłonna, pozwala jednak na dostosowanie instalacji do własnych potrzeb.

Wersja skompilowana rozprowadzana jest w postaci 9,8-megabajtowego archiwum. Archiwum to zawiera zestaw wskazówek co do procedury instalacji, zapisanych w pliku *INSTALL-BINARY*. Są w nim opisane kolejne etapy instalacji podstawowej, składającej się z szeregu poleceń tworzących grupę i użytkownika `mysql`, umieszczających pliki binarne w odpowiednim podkatalogu katalogu */usr/local* oraz ustawiające dla nowoutworzonego użytkownika i grupy uprawnienia dostępu do owego katalogu.

Na początek należy jednak rozpakować archiwum:

```
# tar zxvf mysql-standard-4.0.20-pc-linux-i686.tar.gz
```

Kod źródłowy rozprowadzany jest pod postacią 15,2-megabajtowego archiwum (w formacie SRPM bądź formacie programu `tar`). Po jego pobraniu i rozpakowaniu należy archiwum rozpakować (odpowiednio: poleceniem `rpm` bądź poleceniem `tar`). Archiwum zawiera plik wskazówek odnośnie kompilacji i instalacji oprogramowania, zapisany pod nazwą *INSTALL-BINARY*. Plik zawiera instrukcje dotyczące instalacji „szybkiej", podobne do tych odnoszących się do wersji skompilowanej.

Po zakończeniu wszelkich wymaganych czynności poinstalacyjnych można przejść do katalogu */usr/local/mysql* i uruchomić serwer. Do uruchamiania serwera służy skrypt `mysql_safe` (znajdujący się w katalogu */usr/local/mysql/bin*):

```
# bin/mysqld_safe --user=mysql &
[1] 24798
Starting mysqld daemon with databases from /var/lib/mysql
```

W przypadku instalowania oprogramowania bazy danych MySQL z pakietu RPM, niezbędne wpisy użytkownika i grupy serwera bazy danych MySQL zostaną automatycznie wprowadzone do odpowiednich plików systemowych. W przypadku instalacji samodzielnej należy owe wpisy utworzyć; zarówno użytkownik serwera MySQL, jak i jego

grupa, powinny nosić nazwę `mysql`. Konto użytkownika i grupy `mysql` powinny być skonfigurowane tak, aby zablokować możliwość logowania się w imieniu użytkownika serwera bazy danych — konto `mysql` służy wyłącznie do uruchamiania programu serwera. Tworzeniu kont użytkowników poświęcony był rozdział 9., zatytułowany „Zarządzanie kontami użytkowników".

Inicjalizowanie katalogu danych bazy MySQL

Po zainstalowaniu oprogramowania MySQL należy zainicjalizować **tabelę uprawnień** (ang. *grant table*), czyli tabelę definiującą uprawnienia dostępu do wszystkich baz danych i tabel (a nawet poszczególnych kolumn). Służy do tego program `mysql_install_db`, umieszczony w podkatalogu *scripts* katalogu instalacyjnego bazy danych MySQL. W przypadku instalacji z kodu źródłowego, oprogramowanie bazy danych instalowane jest domyślnie w katalogu */usr/local/mysql*. Program `mysql_install_db` inicjalizuje tabelę uprawnień i tworzy na potrzeby bazy danych konto użytkownika głównego.

Właścicielem katalogu danych MySQL musi być użytkownik, w imieniu którego uruchamiany jest serwer bazy danych MySQL (prawo własności katalogu można zmienić za pomocą polecenia `chown`). Użytkownik ten powinien być również jedynym użytkownikiem posiadającym jakiekolwiek uprawnienia w katalogu (innymi słowy, należy owe uprawnienia zdefiniować za pomocą programu `chmod` jako `700`). Każda inna konfiguracja to luka w zabezpieczeniach i zagrożenie dla spójności bazy danych.

Skrypt `mysql_install_db` powinien być uruchamiany przez użytkownika *root*:

```
# mysql_install_db
Installing all prepared tables
040716 13:35:30  ./bin/mysqld: Shutdown Complete

To start mysqld at boot time you have to copy support-files/mysql.server
to the right place for your system

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
./bin/mysqladmin -u root password 'new-password'
./bin/mysqladmin -u root -h tiger.ck.polsl.gliwice.pl password 'new-password'
See the manual for more instructions.

NOTE:  If you are upgrading from a MySQL <= 3.22.10 you should run
the ./bin/mysql_fix_privilege_tables. Otherwise you will not be
able to use the new GRANT command!

You can start the MySQL daemon with:
cd . ; ./bin/mysqld_safe &

You can test the MySQL daemon with the benchmarks in the 'sql-bench' directory:
cd sql-bench ; perl run-all-tests

Please report any problems with the ./bin/mysqlbug script!

The latest information about MySQL is available on the web at
http://www.mysql.com
Support MySQL by buying support/licenses at https://order.mysql.com
```

Program inicjalizujący przygotowuje bazę danych MySQL do udostępnienia w systemie, wyświetlając przy tym przydatne informacje. Kolejną niezbędną czynnością jest określenie hasła dla głównego użytkownika bazy danych MySQL (patrz punkt następny).

Domyślnie konto użytkownika głównego bazy danych nie posiada hasła. Jest to jedna z pierwszych luk, jakie należy po instalacji oprogramowania serwera bazy danych bezzwłocznie załatać — użytkownik główny bazy danych ma dostęp do wszelkich jej ustawień.

Przypisywanie hasła do konta użytkownika głównego bazy danych MySQL

Aby do konta użytkownika głównego bazy danych przypisać hasło, należy w imieniu tego użytkownika nawiązać połączenie z serwerem bazy danych. Służy do tego polecenie `mysql -u root`. Polecenie to nawiązuje połączenie z serwerem za pośrednictwem klienta MySQL (patrz podrozdział „Program klienta bazy danych MySQL").

```
# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.20-standard

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

mysql>
```

Kiedy konsola wyświetli znak zachęty klienta MySQL, należy wprowadzić następujące polecenie:

```
mysql> SET PASSWORD FOR root = PASSWORD("tajnehasło");
Query OK, 0 rows affected (0.00 sec)
```

Ciąg *tajnehasło* należy rzecz jasna zastąpić własnym hasłem. Polecenie `SET PASSWORD` można też wykorzystywać do przypisywania (i modyfikacji) haseł do kont innych użytkowników bazy danych.

Po wprowadzeniu hasła można opuścić program klienta MySQL; wystarczy w tym celu wpisać polecenie `exit`.

Tworzenie bazy danych

W MySQL bazy danych tworzy się za pośrednictwem zapytania `CREATE DATABASE`. Wcześniej jednak należy nawiązać połączenie z serwerem MySQL, wpisując `mysql -u root -p` i naciskając klawisz *Enter* — polecenie to powoduje nawiązanie połączenia z serwerem w imieniu użytkownika głównego bazy danych; próba nawiązania połączenia zostanie poprzedzona monitem o podanie hasła. Jeśli hasło będzie właściwe, na ekranie ukaże się znak zachęty klienta MySQL. Dopiero wtedy można wprowadzić zapytanie `CREATE DATABASE`. Dla przykładu, prezentowane poniżej zapytanie tworzy bazę danych o nazwie `zwierzeta`:

```
# mysql -u root -p
```

```
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.20-standard

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

mysql> CREATE DATABASE zwierzeta;
Query OK, 1 row affected (0.00 sec)
mysql>
```

Innym sposobem utworzenia bazy danych jest skorzystanie z pośrednictwa programu `mysqladmin`, który udostępnia użytkownikowi głównemu bazy danych opcję `create`, przyjmującą argument w postaci nazwy bazy danych. Dla przykładu, aby utworzyć bazę danych o nazwie `gady`, należałoby skonstruować następujące polecenie:

```
# mysqladmin -u root -p create gady
```

Przyznawanie i odbieranie uprawnień w bazie danych MySQL

Administrator bazy danych będzie zapewne chciał przydzielić sobie pewne **uprawnienia**, których nie będą mieli pozostali użytkownicy bazy danych. Uprawnienia te (ang. *privileges*), mogą być przyznawane i odbierane na czterech różnych poziomach:

- na poziomie globalnym — uprawnienie globalne obejmuje wszystkie bazy danych serwera;
- na poziomie bazy danych — obejmuje wszystkie tabele danej bazy danych;
- na poziomie tabeli — obejmuje wszystkie kolumny danej tabeli w określonej bazie danych;
- na poziomie kolumny — pozwala na dostęp do określonej kolumny w ramach pewnej tabeli i bazy danych.

W rozdziale nie ma miejsca na omówienie wszystkich uprawnień dostępnych w bazie danych MySQL. Ich znaczenie można jednak z należć w dokumentacji bazy danych.

Aby dodać konto użytkownika, należy z serwerem bazy danych nawiązać połączenie w imieniu użytkownika głównego, korzystając z polecenia `mysql -u root -p`. Przed nawiązaniem połączenia wyświetlony zostanie monit o podanie hasła. Dopiero po jego wprowadzeniu klient MySQL wyświetli znak zachęty.

Do przyznawania uprawnień służy zapytanie `GRANT`. Jego składnia jest następująca:

```
GRANT uprawnienie ON obiekt TO użytkownik IDENTIFIED BY 'hasło';
```

`Uprawnienia`, jakie można przyznawać użytkownikom, identyfikowane są słowami kluczowymi. I tak, słowo kluczowe `ALL` oznacza komplet uprawnień.

`Obiekt` określa zasób, którego dotyczy uprawnienie.

`Użytkownik` to nazwa konta użytkownika, który otrzyma uprawnienia.

Hasło to oczywiście hasło, jakie ma zostać przypisane do konta użytkownika. W przypadku, kiedy polecenie dotyczy użytkownika już posiadającego konto opatrzone hasłem, klauzulę IDENTIFIED BY i hasło można pominąć.

Dla przykładu, aby użytkownikowi fefe przyznać komplet uprawnień do bazy danych zwierzeta, należy skonstruować następujące polecenie:

```
GRANT ALL ON zwierzeta.* TO fefe IDENTIFIED BY 'haslofefe';
```

Warto dodać jeszcze wpis:

```
INSERT INTO USER (HOST,USER,PASSWORD) VALUES ('localhost','fefe',Password('haslofefe'));
```

i przeładować:

```
FLUSH PRIVILEGES;
```

Użytkownik fefe będzie mógł po wykonaniu powyższego polecenia nawiązać połączenie z bazą danych zwierzeta, na komputerze lokalnym (localhost), w której będzie dysponował kompletem uprawnień, z możliwością tworzenia i usuwania tabel włącznie. Dla przykładu, użytkownik fefe może zalogować się w bazie danych korzystając z następującego polecenia:

```
$ mysql -h localhost -u fefe -p zwierzeta
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21 to server version: 4.0.20-standard

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

mysql>
```

Dodatkowe opcje wywołania programu klienta MySQL opisane są w podrozdziale „Program klienta bazy danych MySQL”.

Kiedy użytkownikowi fefe nie będą już potrzebne tak rozległe uprawnienia, można je mu odebrać. Służy do tego polecenie REVOKE. Użytkownikowi fefe można odebrać cały pakiet uprawnień naraz:

```
REVOKE ALL ON zwierzeta FROM fefe;
```

Bardziej zaawansowane aspekty zarządzania bazami danych, uprawnieniami i mechanizmami zabezpieczającymi to niestety temat na osobną książkę. Nieco dokumentów na ten temat wymienionych jest w kończącym rozdział o bazach danych podrozdziale „Warto zajrzeć”. Nie zaszkodzi też sięgnąć po jedną z wielu książek poświęconych bazie MySQL, np. *PHP and MySQL Development* [2], autorstwa Luke'a Wellinga i Laury Thompson (wydawnictwo Sams Publishing).

[2] Wydanie polskie: *PHP i MySQL. Tworzenie stron WWW*, Helion, 2002.

Instalowanie i konfigurowanie bazy danych PostgreSQL

Najnowsze wersje oprogramowania serwera baz danych PostgreSQL w postaci kodu źródłowego i pakietów skompilowanych publikowane są na stronach witryny *http://www.postgresql.org/*. Oprogramowanie bazy danych PostgreSQL dystrybuowane jest także w postaci pakietów RPM. Pakietów tych jest kilka; minimalna instalacja powinna obejmować pakiety *postgresql*, *postgresql-server* oraz *postgres-libs*. W razie wątpliwości co do konieczności instalowania pozostałych pakietów warto zapoznać się z treścią pliku */usr/share/doc/postgresql-7.4.2/README.rpm-dist* (plik ten jest tam umieszczany w ramach instalacji pakietu *postgresql*).

W przypadku instalowania oprogramowania serwera PostgreSQL niezbędne do działania serwera konto użytkownika `postgres` (czyli użytkownika, w imieniu którego uruchamiany jest serwer bazy danych PostgreSQL) tworzone jest automatycznie. Można to sprawdzić następującym poleceniem:

```
$ less /etc/passwd | grep postgres
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
```

W przypadku samodzielnej instalacji, czy to z pakietu kodu źródłowego, czy pakietu oprogramowania skompilowanego, czy nawet pakietów RPM innych niż te pochodzące z dystrybucji Fedora, należy konto użytkownika `postgres` utworzyć samodzielnie. Konto owego użytkownika nie powinno dopuszczać logowania; konto to może być dostępne jedynie użytkownikowi uprzywilejowanemu, po wydaniu przez niego polecenia `su` (sposób dodawania do systemu kont nowych użytkowników opisany został w rozdziale 9.). Po utworzeniu konta użytkownika `postgres` można przystąpić do instalowania pobranych pakietów oprogramowania; w przypadku pakietów RPM najlepiej wykorzystać do tego polecenie `rpm -i`.

Inicjalizowanie katalogu danych bazy PostgreSQL

Po pomyślnym zainstalowaniu pakietów RPM należy przystąpić do inicjalizacji katalogu danych. W tym celu należy ów katalog najpierw utworzyć, a do tego potrzebne są uprawnienia użytkownika uprzywilejowanego. W dalszych przykładach niejawnie zakładamy, że katalogiem danych jest */usr/local/pgsql/data/*.

Za pośrednictwem polecenia `mkdir` należy utworzyć katalog */usr/local/pgsql/data* i zmienić prawo własności do katalogu (poleceniami `chgrp` i `chown`) tak, aby jego wyłącznym właścicielem był użytkownik `postgres`. Następnie za pomocą polecenia `su` należy przyjąć uprawnienia użytkownika `postgres` i wykonać w jego imieniu następujące polecenia:

```
# mkdir /usr/local/pgsql
# chown -R postgres /usr/local/pgsql
# chgrp -R postgres /usr/local/pgsql
# su - postgres
-bash-2.05b$ initdb -D /usr/local/pgsql/data
The files belonging to this database system will be owned by user "postgres".
```

```
This user must also own the server process.

The database cluster will be initialized with locale pl_PL.UTF-8.

fixing permissions on existing directory /usr/local/pgsql/data... ok
creating directory /usr/local/pgsql/data/base... ok
creating directory /usr/local/pgsql/data/global... ok
creating directory /usr/local/pgsql/data/pg_xlog... ok
creating directory /usr/local/pgsql/data/pg_clog... ok
selecting default max_connections... 100
selecting default shared_buffers... 1000
creating configuration files... ok
creating template1 database in /usr/local/pgsql/data/base/1... ok
initializing pg_shadow... ok
enabling unlimited row size for system tables... ok
initializing pg_depend... ok
creating system views... ok
loading pg_description... ok
creating conversions... ok
setting privileges on built-in objects... ok
creating information schema... ok
vacuuming database template1... ok
copying template1 to template0... ok

Success. You can now start the database server using:

    /usr/bin/postmaster -D /usr/local/pgsql/data
or
    /usr/bin/pg_ctl -D /usr/local/pgsql/data -l logfile start
```

Powyższe komunikaty informują o pomyślnym zainicjalizowaniu katalogu danych oraz stosownym ustawieniu uprawnień do tego katalogu.

Program `initdb` ustawia uprawnienia dostępu do katalogu danych na `700`. Nie należy tego trybu dostępu zmieniać — każda inna wartość naraża na szwank bezpieczeństwo danych.

Proces nasłuchu serwera bazy danych PostgreSQL, `postmaster`, uruchamia się następującym poleceniem (naturalnie, wciąż w imieniu użytkownika `postgres`):

```
-bash-2.05b$ postmaster -D /usr/local/pgsql/data &
[1] 26196
LOG:  database system was shut down at 2004-07-16 15:22:49 CEST
LOG:  checkpoint record is at 0/9B1058
LOG:  redo record is at 0/9B1058; undo record is at 0/0; shutdown TRUE
LOG:  next transaction ID: 536; next OID: 17142
LOG:  database system is ready
```

W przypadku, kiedy na katalog danych wybrany zostanie katalog inny niż */usr/local/pgsql/data*, należy odpowiednio dostosować wywołanie programu `postmaster`.

W dystrybucji Fedora Core 2 katalog danych serwera bazy danych PostgreSQL to */var/lib/pgsql/data/*. Nie jest to jednak najlepsze miejsce do przechowywania danych, ponieważ w większości systemów katalog */var* montowany jest na partycjach nie dysponujących wystarczającą ilością miejsca na większą ilość danych. W przypadku podjęcia decyzji o zmianie katalogu domyślnego należy stosownie zmodyfikować skrypt rozruchowy serwera PostgreSQL, zapisany w pliku */etc/rc.d/init.d/postgres*.

Tworzenie bazy danych

Tworzenie baz danych dla serwera PostgreSQL nie jest czynnością wielce skomplikowaną. Musi jednak być przeprowadzona przez użytkownika dysponującego uprawnieniami do tworzenia baz danych dla serwera PostgreSQL, początkowo zaś uprawnieniami takimi dysponuje wyłącznie użytkownik `postgres`. Utworzenie bazy danych wymaga w takiej sytuacji wykonania następujących poleceń:

```
# su - postgres
-bash-2.05b$ createdb baza_danych
CREATE DATABASE
```

Gdzie *baza_danych* to nazwa bazy danych do utworzenia.

Program `createdb` jest w rzeczywistości wyłącznie swego rodzaju otoczką programu-klienta `psql`, ułatwiającą tworzenie baz danych przez wyeliminowanie konieczności uruchomienia klienta PostgreSQL. Równie dobrze można jednak utworzyć bazę danych bezpośrednio w programie klienta `psql`, wykonując w nim polecenie `CREATE DATABASE`:

```
CREATE DATABASE baza_danych;
```

Zanim będzie można sensownie wykorzystać program klienta serwera PostgreSQL trzeba utworzyć przynajmniej jedną bazę danych. Bazę tę należy tworzyć, będąc zalogowanym w imieniu użytkownika `postgres`. To zaś można osiągnąć wyłącznie z konta użytkownika uprzywilejowanego, z którego należy wykonać polecenie `su` aby przejąć uprawnienia użytkownika `postgres`. Aby połączyć się z nowoutworzoną bazą danych, należy uruchomić program `psql`, podając w wierszu wywołania nazwę bazy danych:

```
$ psql baza_danych
```

Jeśli w wierszu wywołania programu-klienta serwera PostgreSQL nie zostanie określona nazwa bazy danych, program spróbuje nawiązać połączenie z bazą danych o nazwie identycznej z nazwą użytkownika, który uruchomił program `psql`.

Tworzenie kont użytkowników bazy danych PostgreSQL

W celu utworzenia konta użytkownika bazy danych należy przejąć uprawnienia użytkownika `postgres`. Następnie można skorzystać z polecenia `createuser`, tworzącego konto nowego użytkownika; ów użytkownik może od razu otrzymać uprawnienia dostępu do baz danych i możliwość tworzenia nowych użytkowników bazy danych. Wystarczy twierdząco odpowiedzieć na oba zadane przy okazji tworzenia nowego konta pytania:

```
-bash-2.05b$ createuser bebe
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) y
CREATE USER
```

W powyższym przykładzie użytkownik `bebe` ma zaraz po utworzeniu uprawnienia do tworzenia nowych baz danych oraz kont kolejnych użytkowników (oczywiście z uprawnieniami tymi nie należy przesadzać).

Konto nowego użytkownika można też zainicjalizować z poziomu programu klienta `psql`, uruchamiając go z argumentem określającym bazę danych i wpisując w wierszu poleceń programu zapytanie `CREATE USER`, jak poniżej:

```
CREATE USER bebe;
```

PostgreSQL pozwala na pominięcie w zapytaniu tworzącym użytkownika klauzuli `WITH PASSWORD`. Nowoutworzone konto nie jest wtedy zabezpieczane żadnym hasłem. To poważne naruszenie zasad bezpieczeństwa, więc każdorazowo należy pamiętać o uzupełnieniu zapytania tworzącego konto nowego użytkownika klauzulą `WITH PASSWORD` i ciągiem hasła.

Po zakończeniu pracy program klienta `psql` można zamknąć, wprowadzając polecenie `\q`. Spowoduje to powrót do wiersza poleceń powłoki.

Usuwanie kont użytkowników bazy danych PostgreSQL

Do usuwania kont użytkownika bazy danych PostgreSQL służy polecenie `dropuser`, uzupełnione nazwą konta użytkownika:

```
-bash-2.05b$ dropuser bebe
DROP USER
```

Można też — korzystając z pośrednictwa programu `psql` — zalogować się do serwera bazy danych i wykonać zapytanie `DROP USER`. Oto przykład:

```
-bash-2.05b$ psql demodb
Welcome to psql 7.4.2, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

baza=# DROP USER bebe;
DROP USER
baza=# \q
-bash-2.05b$
```

Przyznawanie i odbieranie uprawnień użytkownikom bazy danych PostgreSQL

Podobnie, jak w bazie danych MySQL, przyznawanie i odbieranie uprawnień w bazie danych PostgreSQL przeprowadza się za pośrednictwem zapytań `GRANT` oraz `REVOKE`. Składnia zapytania jest tu nieco inna niż w przypadku MySQL, jako że PostgreSQL nie uwzględnia klauzuli `IDENTIFIED BY` — użytkownicy baz danych PostgreSQL otrzymują hasła w ramach procedury tworzenia konta użytkownika (`CREATE USER`). Oto składnia zapytania `GRANT`:

```
GRANT uprawnienie ON obiekt TO użytkownik;
```

Dla przykładu, poniższe polecenie przyznaje użytkownikowi `fefe` wszelkie uprawnienia w ramach bazy danych `baza`:

```
GRANT ALL ON DATABASEbaza TO fefe;
```

Do odbierania uprawnień służy zapytanie `REVOKE`. Oto przykład jego zastosowania:

```
REVOKE ALL ON DATABASE baza FROM fefe;
```

Powyższe polecenie odbiera użytkownikowi `fefe` wszystkie uprawnienia, jakimi dysponował on w ramach bazy danych `baza`.

Zaawansowane zagadnienia administracji bazą danych to kwestie bardzo złożone. Nie sposób ująć ich wszystkich w jednym rozdziale, którego zadaniem nie było zresztą tylko wprowadzenie Czytelników w zagadnienia administracji bazami danych. Więcej informacji na temat administrowania bazami danych PostgreSQL znaleźć można w dokumentacji bazy danych oraz w licznych książkach, poświęconych bazie PostgreSQL w całości (jak choćby *PostgreSQL*[3] wydawnictwa Sams Publishing).

Programy-klienty baz danych

Zarówno w bazie danych MySQL, jak i PostgreSQL, system zarządzania bazą danych implementowany jest w architekturze klient-serwer. Najprościej rzecz ujmując, serwer bazy danych obsługuje żądania dostępu do bazy danych; żądania te są przesyłane przez program klienta, który również odbiera odpowiedzi serwera i prezentuje otrzymywane zbiory wynikowe użytkownikowi.

Użytkownik nigdy nie komunikuje się bezpośrednio z serwerem bazy danych, nawet, jeśli pracuje na tym samym węźle, na którym uruchomiony jest proces serwera. We wszystkich odwołaniach do bazy danych pośredniczy program klienta; program ten może być uruchomiony równie dobrze na komputerze, na którym działa serwer, jak i łączyć się z tym komputerem (i procesem serwera) z drugiego końca świata.

Użytkownicy obu omawianych systemów zarządzania bazami danych mają do dyspozycji programy-klienty, udostępniające interfejs wierszowy. To bardzo prymitywny i z tego względu rzadko wykorzystywany przez użytkowników końcowych sposób komunikowania się z bazą danych. Administrator bazy danych nie ma jednak zazwyczaj wyboru, ponieważ tylko to prymitywne narzędzie jest na tyle elastyczne, że pozwala na testowanie nowych zapytań bez konieczności osadzania ich w kodzie aplikacji użytkowych i konstruowania ich interfejsu. W kolejnych częściach niniejszego rozdziału Czytelnik będzie miał jednak okazję poznać klienta bazy danych MySQL działającego w trybie graficznym, jak również szereg interfejsów WWW, dostępnych i dla baz MySQL, i baz danych PostgreSQL.

[3] Albo jednej z poświęconych tej bazie danych książek, wydanych po polski: *PostgreSQL. Praktyczny przewodnik* (Helion, 2002) czy *Bazy danych i PostgreSQL. Od podstaw* (Helion, 2002).

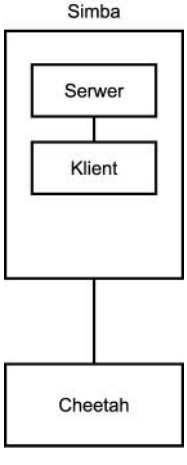
W kolejnych punktach przetestowane zostaną dwie najpopularniejsze metody komunikacji ze zdalnymi bazami danych, metoda odwoływania się do serwera lokalnego oraz koncepcja korzystania z bazy danych za pośrednictwem interfejsu WWW.

Konfigurując bazę danych warto starannie przemyśleć kwestie uprawnień dostępu do bazy danych. Należy się przy tym zastanowić, czy użytkownicy powinni mieć możliwość tworzenia i usuwania baz danych, czy może powinni zostać ograniczeni do korzystania wyłącznie z tych baz, które zastaną. Nie wszyscy użytkownicy powinni mieć możliwość modyfikowania rekordów poszczególnych baz danych — część z nich powinna zadowolić się dostępem do danych w trybie do odczytu. A co z tzw. resztą świata? Przecież użytkownicy z zewnątrz, łączący się za pośrednictwem sieci Internet, nie powinni być może mieć żadnego dostępu do baz danych? Wszystkie te wątpliwości spadają na głowę administratora bazy danych.

Dostęp do bazy danych za pośrednictwem SSH

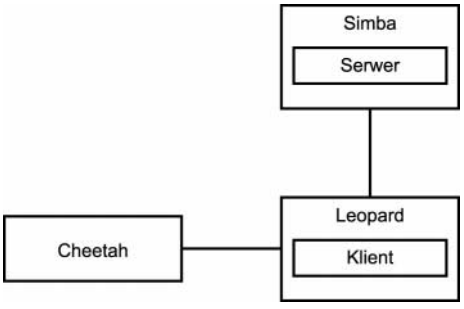
W tym punkcie przedstawione zostaną dwa często spotykane tryby dostępu do zdalnych serwerów baz danych. W pierwszym z nich użytkownik loguje się do systemu serwera bazy danych, korzystając z pośrednictwa SSH (aby zapewnić bezpieczeństwo transmisji pomiędzy serwerem bazy danych a węzłem lokalnym). Następnie, już „lokalnie", czyli na węźle, na którym działa serwer bazy danych uruchamia program klienta. Taki schemat ilustrowany jest rysunkiem 17.4.

Rysunek 17.4. *Użytkownik loguje się do systemu serwera bazy danych, uruchomionego na węźle Simba, samemu zasiadając przed monitorem komputera Cheetah; program klienta bazy danych uruchamiany jest na węźle Simba*



Metoda druga, ilustrowana rysunkiem 17.5, zakłada, że użytkownik loguje się (za pośrednictwem bezpiecznej powłoki SSH) na węźle zdalnym i na nim uruchamia program klienta bazy danych. Serwer jednak w tym scenariuszu działa na jeszcze innym węźle. W takiej konfiguracji w przetwarzanie zaangażowane są trzy węzły: węzeł, przed którym zasiada użytkownik, węzeł programu klienta bazy danych oraz węzeł serwera bazy danych.

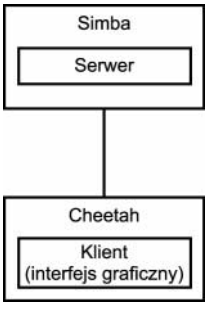
Najważniejszym elementem schematu z rysunku 17.5 jest pośredniczący w komunikacji pomiędzy użytkownikiem a serwerem węzeł Leopard. W tym układzie program klienta bazy danych nie jest już uruchamiany na węźle serwera; nie jest też jednak uruchamiany lokalnie, w systemie użytkownika.



Rysunek 17.5. *Użytkownik loguje się z węzła Cheetah na węzeł Leopard i uruchamia na nim program klienta bazy danych; dopiero ten nawiązuje połączenie z węzłem serwera bazy danych (Simba)*

Dostęp do serwera bazy danych za pośrednictwem programu klienta wyposażonego w interfejs graficzny

Użytkownik może zalogować się do serwera bazy danych za pośrednictwem klienta działającego w trybie graficznym; klient ten może działać w systemie Windows, Macintosh czy na stacji roboczej UNIX. Program klienta nawiązuje połączenie z serwerem bazy danych. W takim przypadku program klienta działa lokalnie na węźle, przed którym zasiada użytkownik (patrz rysunek 17.6).

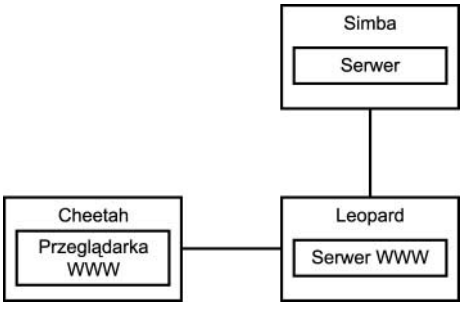


Rysunek 17.6. *Użytkownik uruchamia program klienta wyposażony w interfejs graficzny na węźle Cheetah; program ten nawiązuje połączenie z serwerem bazy danych, uruchomionym na węźle Simba*

Dostęp do serwera bazy danych za pośrednictwem interfejsu WWW

W tym punkcie przyjrzymy się dwóm prostym przykładom odwoływania się do serwera bazy danych za pośrednictwem odpowiednio spreparowanej strony WWW. W pierwszym przykładzie użytkownik korzysta z zawartości bazy danych, posługując się zbiorem formularzy publikowanych w sieci WWW. Na pierwszy rzut oka zdaje się, że w układzie tym program klienta uruchomiony jest na węźle użytkownika. Nie jest to jednak zgodne ze stanem faktycznym, gdyż program klienta obsługiwany jest przez serwer WWW. Przeglądarka, którą użytkownik uruchomił lokalnie, prezentuje jedynie wynik działania programu klienta i pozwala na przekazywanie do niego danych. Właściwe oprogramowanie klienta składa się tu z serwera WWW (obsługującego żądania nadsyłane przez przeglądarkę) oraz skryptu CGI, serwletu w języku Java czy osadzanego w dokumencie HTML skryptu języka PHP czy JSP.

W kontekście baz danych pojęcia interfejsu i klienta stosuje się często zamiennie. Jednak, co ujawnia rysunek 17.7, klient i interfejs to w omawianym przypadku dwa różne elementy. Interfejsem jest tutaj formularz bądź formularze, prezentowane użytkownikowi przez przeglądarkę WWW. Klient nosi zaś w tym układzie nazwę **warstwy pośredniczącej**.



Rysunek 17.7. *Użytkownik korzysta z bazy danych za pośrednictwem sieci WWW; interfejsem jest tu przeglądarka WWW użytkownika a klientem oprogramowanie działające na węźle Leopard; na węźle Simba działa serwer bazy danych*

Podział funkcjonalny może jednak w przypadku dostępu za pośrednictwem interfejsu WWW wyglądać nieco inaczej. Klient może być bowiem aplikacją dwuczęściową, przy czym jedna z tych części działa na komputerze użytkownika, druga zaś jest obsługiwana przez serwer WWW. Dla przykładu, programista aplikacji baz danych może uzupełnić formularz prezentowany w przeglądarce o kod sprawdzający poprawność wprowadzonych do formularza danych. W takiej sytuacji zapytanie jest częściowo przetwarzane w systemie użytkownika, a częściowo tylko w serwerze WWW (często do węzła użytkownika oddelegowywane są operacje kontroli błędów; pozwala to na zmniejszenie obciążenia serwera i natężenia ruchu sieciowego — zapytanie jest sprawdzane pod kątem potencjalnych błędów jeszcze przed przesłaniem go do serwera bazy danych).

Program klienta bazy danych MySQL

Plik wykonywalny programu wierszowego klienta bazy danych MySQL nosi nazwę `mysql`. Program wywołuje się następująco:

```
mysql [opcje] [nazwa bazy danych]
```

Niektóre z opcji rozpoznawanych przez program `mysql` wymienione zostały w tabeli 17.1. Nazwa bazy danych to (podobnie, jak opcje) argument opcjonalny — w przypadku jego braku program zostanie uruchomiony bez nawiązywania połączenia z bazą danych.

Tabela 17.1. *Wybrane opcje wywołania programu mysql*

Opcja	Działanie
`-h` *nazwa_węzła*	Połączenie z serwerem uruchomionym na węźle zdalnym *nazwa_węzła*
`-u` *nazwa_użytkownika*	Połączenie w imieniu użytkownika *nazwa_użytkownika*
`-p`	Wymuszenie zapytania o hasło. Opcja wymagana wtedy, kiedy konto użytkownika, w imieniu którego nawiązywane jest połączenie, zabezpieczane jest hasłem
`-P` *numer_portu*	Numer portu, z którym program powinien nawiązać połączenie
`-?`	Wyświetlenie komunikatu pomocy

Program rozpoznaje znacznie więcej opcji, ale te zaprezentowane w tabeli 17.1 są wykorzystywane najczęściej. Pełny wykaz dostępnych opcji wraz z ich znaczeniem można obejrzeć na stronie podręcznika systemowego `man` pod hasłem `mysql`.

Hasło użytkownika, w imieniu którego nawiązywane jest połączenie, można określić w wierszu wywołania programu `mysql`, umieszczając je za przełącznikiem `-p`. Nie trzeba wtedy czekać na prośbę o podanie hasła. Jednak taki sposób wywoływania programu klienta `mysql` jest bardzo niebezpieczny. Hasło będzie bowiem włączone do ciągu wywołania polecenia, co oznacza, że można je będzie odczytać z listy procesów systemu. To poważne zagrożenie dla bezpieczeństwa danych; warto więc przyjąć zasadę, aby nigdy nie przekazywać hasła użytkownika bazy danych w wierszu wywołania programu `mysql`.

Program można uruchomić bez wskazania docelowej bazy danych. W takim przypadku warto po uruchomieniu programu skorzystać z polecenia `help` — program wyświetli obsługiwane polecenia. Wśród nich jest polecenie nawiązania połączenia z bazą danych:

```
mysql> help

MySQL commands:
Note that all text commands must be first on line and end with ';'
help    (\h)    Display this help.
?       (\?)    Synonym for `help'.
clear   (\c)    Clear command.
connect (\r)    Reconnect to the server. Optional arguments are db and host.
edit    (\e)    Edit command with $EDITOR.
ego     (\G)    Send command to mysql server, display result vertically.
exit    (\q)    Exit mysql. Same as quit.
go      (\g)    Send command to mysql server.
nopager (\n)    Disable pager, print to stdout.
notee   (\t)    Don't write into outfile.
pager   (\P)    Set PAGER [to_pager]. Print the query results via PAGER.
print   (\p)    Print current command.
quit    (\q)    Quit mysql.
rehash  (\#)    Rebuild completion hash.
source  (\.)    Execute a SQL script file. Takes a file name as an argument.
status  (\s)    Get status information from the server.
tee     (\T)    Set outfile [to_outfile]. Append everything into given outfile.
use     (\u)    Use another database. Takes database name as argument.
```

Ostatnie z prezentowanych poleceń, `use`, pozwala na określenie bazy danych, z którą program ma się połączyć (np. `zwierzeta`). Trzeba pamiętać, że połączenie uda się tylko wtedy, kiedy bieżący użytkownik programu `mysql` ma uprawnienia dostępu do wskazanej bazy danych:

```
mysql> use zwierzeta
Database changed
mysql>
```

Program klienta bazy danych PostgreSQL

Program klienta bazy danych PostgreSQL, oferujący wierszowy interfejs poleceń, nosi nazwę `psql`. Podobnie, jak to ma miejsce w przypadku programu `mysql`, program `psql` może zostać uruchomiony z argumentem wskazującym docelową bazę danych. I również `psql` rozpoznaje szereg opcji, z których najczęściej wykorzystywane zostały umieszczone w tabeli 17.2.

Tabela 17.2. *Wybrane opcje wywołania programu psql*

Opcja	Działanie
`-h` *`nazwa_węzła`*	Połączenie z serwerem uruchomionym na węźle zdalnym *`nazwa_węzła`*
`-p` *`numer_portu`*	Numer portu, z którym program powinien nawiązać połączenie
`-U` *`nazwa_użytkownika`*	Połączenie w imieniu użytkownika *`nazwa_użytkownika`*
`-W`	Wymuszenie zapytania o hasło. W wersjach bazy danych PostgreSQL nowszych niż wersja 7 zapytanie o hasło jest generowane automatycznie, na żądanie serwera
`-?`	Wyświetlenie komunikatu pomocy

Pełny wykaz dostępnych opcji wraz z ich znaczeniem można obejrzeć na stronie podręcznika systemowego `man` pod hasłem `psql`.

Interfejsy graficzne

Czytelnicy zainteresowani dostępem do serwera baz danych za pośrednictwem wygodniejszego, bo działającego w trybie graficznym, programu klienta, mogą odetchnąć z ulgą — dostęp taki jest możliwy.

Otóż dla bazy danych MySQL dostępny jest oficjalny klient działający w trybie graficznym: MySQLGUI. MySQLGUI dostępny jest w postaci kodu źródłowego i w wersjach skompilowanych w witrynie MySQL, publikowanej pod adresem *http://www.mysql.com/*.

Dla baz danych MySQL i PostgreSQL dostępne są też interfejsy implementowane za pośrednictwem stron WWW. Wśród takich interfejsów można wymienić phpMyAdmin oraz phpPgAdmin. Oba te produkty korzystają z osadzanego w formularzach HTML języka skryptowego PHP, wymagają więc zainstalowania dla serwera WWW modułu obsługi języka PHP. Nie obejdzie się też, rzecz jasna, bez zainstalowania samego serwera WWW.

Przydatne polecenia systemu Linux

Przy tworzeniu i manipulowaniu bazami danych w dystrybucji Fedora pomocne mogą się okazać następujące polecenia:

- `createdb` — polecenie tworzące bazę danych PostgreSQL;
- `createuser` — polecenie tworzące konto użytkownika bazy danych PostgreSQL;
- `dropdb` — polecenie usuwające bazę danych PostgreSQL;
- `dropuser` — polecenie usuwające konto użytkownika bazy danych PostgreSQL;
- `mysql` — interaktywny program klienta bazy danych MySQL;
- `mysqladmin` — polecenie administrujące zasobami bazy danych MySQL;
- `mysqldump` — program wykonujący zrzut zawartości bazy danych bądź tabeli MySQL;
- `pg_ctl` — narzędzie służące do uruchamiania, zatrzymywania i przeładowywania serwera PostgreSQL;
- `psql` — interaktywny program klienta bazy danych PostgreSQL.

Warto zajrzeć

Oto lista witryn, które warto odwiedzić w poszukiwaniu informacji o bazach danych opisywanych w niniejszym rozdziale:

- *http://www.mysql.com/* — oficjalna witryna WWW oprogramowania serwera baz danych MySQL. Można tu znaleźć najnowsze wersje oprogramowania, jak również aktualne wersje dokumentacji bazy danych MySQL. Tutaj można też zamówić usługę wsparcia technicznego, przydatnego użytkownikom wykorzystującym bazę danych MySQL w środowisku korporacyjnym (wielu użytkowników korporacyjnych nie spojrzy nawet na oprogramowanie, dla którego nie można uzyskać pomocy technicznej).
- *http://www.postgresql.org/* — oficjalna witryna projektu serwera baz danych PostgreSQL. Tutaj publikowane są powiadomienia o najnowszych wydarzeniach związanych z projektem, tu należy szukać najnowszych wersji oprogramowania i dokumentacji.
- *http://www.postgresql.org/docs/7.3/interactive/tutorial.html* — wprowadzenie do korzystania z serwera PostgreSQL; znakomita, dostępna on-line dokumentacja dla wszystkich chcących rozpocząć pracę z bazami danych PostgreSQL.
- *http://www.pgsql.com/* — witryna firmy, oferującej płatne wsparcie techniczne dla użytkowników baz danych PostgreSQL.