Flash MX. Kompendium programisty

Autor: P.S. Woods
Tłumaczenie: Ryszad Glegoła
ISBN: 83-7197-957-6
Tytuł oryginału: Macromedia Flash MX Developer's Guide
Format: B5, stron: 424

Trzeba przyznać, że gdy pojawił się Flash 5, wydawać się mogło, że następne kilka lat minie na powolnym dostosowywaniu się do przełomowych możliwości tej wersji. Okazało się jednak, że nowe wydanie tego programu jest równie fascynujące — i niesie ze sobą tyle samo zmian, co poprzednie.

Istnieje kilka oczywistych innowacji wersji MX, jak na przykład ogromnie rozbudowane słownictwo języka ActionScript oraz nowe opcje wideo. Jest jednak kilka innych nowości, które nie rzucają się w oczy aż tak bardzo, jak choćby fakt, że przybywa nowych i coraz lepszych, a jednocześnie coraz tańszych narzędzi do tworzenia animacji wektorowej w technologii Flash.

Główna część tej książki poświęcona jest skryptom i budowaniu aplikacji Flash, począwszy od najprostszych skryptów JavaScript, współpracujących z interfejsem użytkownika stworzonym w programie Flash, aż po aplikacje, w których za takim interfejsem kryje się zaplecze oparte na bazie danych MySQL i pośredniczącymi między nią a interfejsem skryptami PHP. Umiejętność pisania skryptów jest chyba najbardziej istotną umiejętnością użytkownika programu Flash – trudno bez niej zrobić cokolwiek poza klatkową animacją. Dlatego też skrypty zostały w tej książce omówione bardzo szczegółowo.

Kto powinien przeczytać tę książkę?
Każdy spec od marketingu będzie próbował wmówić człowiekowi, że książka, którą akurat sprzedaje, jest o wszystkim i dla wszystkich. Ten chwyt reklamowy nigdy jeszcze chyba nikogo nie przekonał — tak czy owak, ta książka z pewnością nie jest dla każdego użytkownika programu Flash, jaki stąpa po tej planecie. Na przykład, jeżeli interesuje nas programowanie zorientowane obiektowo i szukamy książki z przepisami na wiele niestandardowych klas — szukajmy dalej. W tej książce jest zaledwie jeden jedyny rozdział poświęcony programowaniu obiektowemu, rozpoczynający się od podstawowych zagadnień i omawiający kilka pouczających przykładów (rozdział 2.).

Z drugiej strony, autor włożył wiele pracy w omówienie tematów, które są poszukiwane najbardziej i o których najmniej można się dowiedzieć, poprzestając jedynie na materiałach udostępnianych w Internecie przez innych użytkowników programu Flash. Jeżeli przeczytaliśmy dokumentację programu Flash MX i chcemy zagłębić się jeszcze bardziej w dowolny jego aspekt, najprawdopodobniej znajdziemy tu dla siebie wiele użytecznych informacji. Nawet ci użytkownicy, którzy w materii skryptów czują się bardzo pewnie, znajdą tu rozdziały skutecznie uzupełniające ich wiedzę.

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

Spis treści

12.

MySQL

W tym rozdziale zapoznamy się pobieżnie z cudownym światem MySQL, gdzie wszystko jest szybkie, bezpłatne i pełne użytecznych funkcji. Teoria oraz przykłady zawarte w tym rozdziale dostarczą nam wystarczająco dużo wiedzy, aby móc stworzyć i uruchomić naszą pierwszą aplikację Flash opartą na bazie danych.

Przykłady w tym rozdziale zostały zbudowane wokół fikcyjnej winiarni *PSWoods Vineyards*. Zbudujemy w programie Flash interfejs użytkownika pozwalający odczytywać pobierane z bazy danych specjalności miesiąca; zbudujemy aplikację do zarządzania tą bazą danych oraz aplikację wysyłającą pocztówki — wszystkie te elementy będą miały interfejs użytkownika opracowany w całości w technologii Flash.

W niniejszym rozdziale można wyodrębnić trzy główne części. Pierwsza to obowiązkowe wprowadzenie w temat i kilka słów odnośnie historii relacyjnych baz danych. Następna część to przyspieszony kurs języka zapytań SQL oraz omówienie konfiguracji i podstawowych funkcji serwera baz MySQL. Ostatnią część rozdziału poświęcimy przykładom praktycznym — zaczniemy od zwykłego, klasycznego rozwiązania opartego na HTML, a następnie przejdziemy do kilku interesujących aplikacji WWW, łączących w sobie PHP, MySQL i Flash.

Pliki przykładów zostały tak opracowane, aby można było skopiować cały folder *przyklady/* z katalogu bieżącego rozdziału i umieścić go w katalogu *htdocs* lokalnego serwera Apache. Wystarczy skopiować cały folder, zmienić prawa odczytu i zapisu plików (zaznaczamy pliki, klikamy prawym przyciskiem myszy, wybieramy *Właściwości* i usuwamy zaznaczenie z pola wyboru przy opcji *Tylko do odczytu*) i wszystko powinno działać. Nie zapomnijmy uruchomić bazy MySQL, a następnie serwera Apache — w przeciwnym razie przykłady nie będą działać.

Trochę historii

Aby w pełni docenić, jak wielkim błogosławieństwem dla projektanta WWW jest MySQL, trzeba najpierw poznać odrobinę historii. Pomysł indeksowania danych jest zapewne tak stary, jak techniki ich zapisu. Istnieją setki, jeżeli nie tysiące prac naukowych, książek i artykułów w Internecie, które przypisują wielu różnym historycznym systemom miano „pierwszej bazy danych" Nie skorzystamy tu zbyt wiele, roztrząsając, jaki wynalazek sprzed ery komputerów *naprawdę* można byłoby nazwać pierwszą bazą danych; ciekawostką jest jednak ewidentnie zakorzenione w ludzkiej naturze dążenie do porządkowania zebranych informacji.

Jako głos w dyskusji na temat pierwszej bazy danych z pewnością można rozważyć stwierdzenie, że był nią wynalazek niejakiego Hermana Holleritha, amerykańskiego Niemca osiadłego od dwóch pokoleń w Stanach Zjednoczonych. Pracował on pod koniec XIX w. dla amerykańskiego Biura Spisowego. Maszyna Holleritha na bieżąco zliczała na dziurkowanych kartach kolumny zaznaczone w poszczególnych wierszach. Chociaż sam termin pojawił się później, wynalazek Holleritha, zbudowany z funduszy amerykańskiego Biura Spisowego i używany później do automatycznej obróbki danych, był fundamentem, na którym *pod koniec XIX wieku* został zbudowany amerykański olbrzym informatyczny, firma IBM. Całą tę historię, wraz ze zdjęciami pierwszych maszyn liczących, można znaleźć pod adresem *www-1.ibm.com/ibm/history/history/decade_1890.html*.

Termin „baza danych" powstał około roku 1960, w momencie narodzin elektronicznych maszyn liczących praktycznego użytku. Bazy danych stanowiły wtedy nie lada sensację, choć były wówczas, rzecz jasna — według dzisiejszych standardów — śmiesznie małe i powolne.

Swoistą rewolucją w historii rozwoju baz danych było zastosowanie napędu dyskowego, gwarantującego swobodny i natychmiastowy dostęp do danych, co nie było możliwe w przypadku jego poprzednika — napędu taśmowego. Dysk pozwalał maszynie odczytującej dane w dowolnej chwili przejść do obszaru na jego powierzchni, w którym zapisane były potrzebne dane — nie trzeba było już czekać, aż przewinie się ogromna szpula taśmy magnetycznej.

W tamtych czasach szczytem zaawansowania dla aplikacji opartej na bazie danych była tzw. *nawigacyjna baza danych* — zbiory danych wymagające od użytkownika znajomości miejsca, w którym zapisane były interesujące go dane. Oznaczało to, że korzystanie z takiej bazy danych wymagało nawigowania po jej hierarchii za pomocą języka niskiego poziomu.

Nawigacyjne bazy danych uważano wówczas za szczyt możliwości tej techniki i nie sądzono, że można w tej dziedzinie wymyślić cokolwiek nowego. Nic więc dziwnego, że przełomowa praca E. F. Codda z roku 1970 pt. „A Relational Model of Data for Large Shared Data Banks" („Relacyjny model danych w dużych, współużytkowanych bazach danych") nie spotkała się ze zbyt ciepłym przyjęciem ze strony pracodawcy autora, firmy IBM. Zawarte w tym opracowaniu pomysły, w tym postulat użycia języka wysokiego poziomu do operacji na danych oraz brak potrzeby znajomości dokładnej hierarchii danych przez programistę (użytkownika), stały się podstawowymi założeniami baz relacyjnych w dzisiejszym wydaniu. Należy do nich np. działająca na lokalnym komputerze użytkownika baza Microsoft Access, a także obsługiwana przez serwer baza Oracle.

Serwer baz danych MySQL powstał dzięki pracy Michaela „Monty" Wideniusa, ówczesnego pracownika szwedzkiej firmy TcX. Chociaż stworzenie wysokiej klasy systemu bazodanowego w warunkach domowych nie jest niczym niemożliwym, MySQL był rozwiązaniem szczególnym — został przystosowany tak, aby mógł go pobrać, skompilować i używać każdy, kto dysponuje dostępem do Internetu.

We współczesnym przemyśle bazodanowym, generującym miliardy dolarów zysku rocznie, MySQL jako serwer relacyjnych baz danych o wysokiej wydajności dostępny całkowicie za darmo jest przypadkiem odosobnionym. Jak wiadomo — co można wywnioskować na podstawie obecności „głośnych" nazw w historii baz danych — aplikacje bazodanowe klasy przemysłowej zawsze były bardzo kosztowne. Serwer baz danych MySQL, w połączeniu z również bezpłatnym środowiskiem jego działania (Unix, Linux i pokrewne systemy), jest wobec tego naprawdę atrakcyjny dla każdego, kto potrzebuje taniego, szybkiego i funkcjonalnego rozwiązania tego typu. W połączeniu z PHP i Apache, MySQL uznawany jest przez wielu (także przez firmę Netcraft, utrzymującą się z badania zaplecza serwisów WWW) za standardowy zestaw aplikacji WWW dla serwerów WWW działających w systemie Linux.

Co oferuje MySQL?

Użytkownicy baz danych znani są z ciągłego porównywania różnych rzeczy — nie wyłączając też samych baz danych. Serwer MySQL często porównywany jest zarówno z komercyjnymi, jak i z innymi bezpłatnymi bądź udostępnianymi jako *open source* bazami danych, np. PostgreSQL. Bez zagłębiania się w testy porównawcze i inne szczegóły, jakie mogą zainteresować tylko specjalistów od baz danych, rzućmy okiem na kilka najważniejszych kryteriów porównawczych.

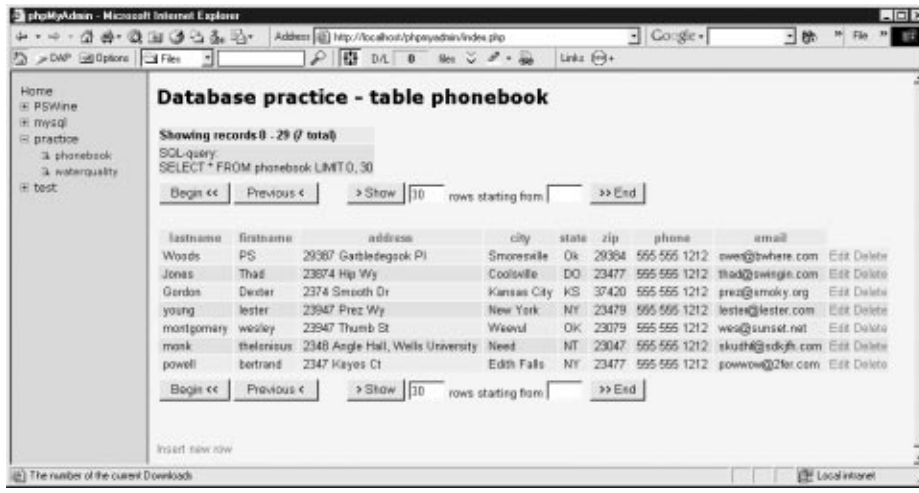
- *Osiągi.* Pod względem prędkości MySQL wypada korzystnie w porównaniu z jakąkolwiek inną, komercyjną lub darmową bazą danych. Choć zapewne nie można stwierdzić, która ze spotykanych w Sieci baz danych jest najszybsza, bazy MySQL ogólnie uważa się za szybsze od najbardziej znanego przedstawiciela ich bezpłatnej konkurencji, bazy PostgreSQL. Twórcy MySQL opublikowali w Internecie wyniki przeprowadzonego przez nich testu osiągów tej bazy — są one do wglądu pod adresem *www.mysql.com/information/benchmarks.html*.
- *Stabilność.* Jeżeli korzystamy z bazy danych tylko dla potrzeb zarządzania treścią naszego serwisu WWW lub innych, podobnych aplikacji służących do obsługi danych *będących* najważniejszą częścią prowadzonej przez nas firmy, prawdopodobnie nigdy nie doświadczymy problemów ze stabilnością baz MySQL — ani na roboczym serwerze lokalnym, jakiego używamy w tej części książki, ani na docelowym serwerze internetowym. Należy mieć jednak świadomość, że komercyjne pakiety bazodanowe, np. Oracle, posiadają ulepszenia (określane zbiorczą nazwą ACID), których brakuje bazom MySQL. Ulepszenia te gwarantują w mniejszym lub większym stopniu odporność tych baz na usterki, w tym także na awarie sprzętowe. Pełny opis tego zagadnienia znajduje się pod adresami: *www.philip.greenspun.com/wtr/aolserver/introduction-2.html* oraz *www.openacs.org/philosophy/why-not-mysql.html*.

- *Zgodność ze standardem SQL.* Także i na tym polu MySQL spisuje się poprawnie. Wygodną cechą połączenia PHP i MySQL jest łatwość takiego przekształcenia skryptu PHP, aby obsługiwał inny typ serwera bazy danych. Przeważnie wystarczy zmiana przedrostka w nazwach funkcji PHP obsługujących bazę danych. Także odwrotnie — system pojęć i składnia każdej dobrej bazy SQL działać będzie w MySQL.
- *Interfejsy ze skryptami działającymi po stronie serwera.* Jeżeli przydały nam się wiadomości zawarte w tej części książki, nigdy zapewne nie spotkamy języka programowania lub języka skryptowego przeznaczonego dla zastosowań WWW, który nie obsługiwałby baz MySQL. Obsługują je takie języki, jak PHP, Perl, Java, Visual Basic czy Visual C++.
- *Bogactwo zastosowań.* Owszem, MySQL posiada mniejszy zestaw funkcji niż Oracle, czy nawet PostgreSQL, jednak świetnie nadaje się dla większości zastosowań potrzebnych użytkownikom programu Flash. Fakt, że serwer MySQL nie specjalizuje się w tak zaawansowanych dziedzinach jak transakcje odporne na zakłócenia (co zwykłym śmiertelnikom nie jest specjalnie przydatne), sprawia, iż pochłania on o wiele mniej zasobów systemowych niż inne serwery baz danych. To zaś jest jak najbardziej dobra wiadomość — zwłaszcza że zamierzamy przecież stworzyć kopię docelowego serwera WWW na naszym własnym, lokalnym sprzęcie.
- *Łatwość instalacji i użycia.* Jak zauważymy, czytając następny fragment tego rozdziału, MySQL instaluje się w systemie Windows w miarę automatycznie. Doświadczonemu administratorowi łatwo też go skompilować i zainstalować na uniksowym czy linuksowym serwerze obsługiwanym przez Apache. Na dodatek, komponent bazy danych MySQL działający po stronie klienta jest nieskomplikowany i łatwy w użyciu.

Model klient–serwer MySQL

Oprogramowanie bazodanowe MySQL składa się z dwóch podstawowych komponentów: serwera i klienta. Serwer to część aplikacji wykonująca całą pracę związaną z jej obsługą, tworzeniem i modyfikowaniem danych. Jest to jedyna część, która działa, gdy udostępniane przez nią bazy danych nie są akurat modyfikowane za pomocą klienta. Także skrypty serwerowe mogą porozumiewać się z serwerem MySQL i wydobywać dane z baz, nie używając klienta.

Klient to dostępna z poziomu wiersza poleceń aplikacja służąca do tworzenia i edycji baz danych; jest to ich domyślny interfejs. Istnieje szereg skryptów PHP mogących pełnić rolę klienta, zwykle umożliwiając wprowadzanie danych i wysyłanie zapytań wizualnie, np. za pośrednictwem interfejsu w postaci strony WWW. Jedną z takich aplikacji jest PHPMyAdmin, instalowany w katalogu *htdocs* serwera Apache w przypadku, gdy ten ostatni został zainstalowany jako część pakietu PHPTriad. Na rysunku 12.1 przedstawiony jest interfejs aplikacji PHPMyAdmin (aplikacja PHPMyAdmin umożliwia również wybór języka interfejsu. Pośród dostępnych języków znajduje się także język polski — *przyp. tłum.*).



Rysunek 12.1. *Interfejs aplikacji PHPMyAdmin, zbudowanego w PHP klienta baz danych MySQL*

Innym ciekawym rozwiązaniem w tej dziedzinie jest graficzny interfejs użytkownika bazy MySQL. Został on stworzony jako alternatywa dla tradycyjnego klienta obsługiwanego z poziomu wiersza poleceń, z którego będziemy korzystać w tym rozdziale. W czasie pisania tej książki nie była to jeszcze aplikacja klasy Microsoft Access, mimo to daje ona łatwy dostęp do poleceń wprowadzanych w trybie tekstowym, na przykład pozwala tworzyć tabele i wprowadzać zapytania SQL. Oferuje też dodatkowo nowe cechy — jak choćby możliwość bezpośredniej edycji obiektów w poszczególnych tabelach, jak w przypadku lokalnych baz danych typu Access.

Instalacja

Instalacja baz danych MySQL w dowolnej wersji systemu Windows sprowadza się do uruchomienia programu instalacyjnego. Cały proces instalacyjny sprowadza się właściwie do rozpakowania skompilowanego programu do wybranego przez użytkownika katalogu. Najnowszy komponent pakietu MySQL, wspomniany interfejs graficzny, instalowany jest w podobnie prosty sposób.

Możemy też korzystać z MySQL jako z jednej z usług serwera NT/2000, nie zmieniając domyślnego dla tego systemu serwera WWW. Nie będziemy omawiać tutaj przebiegu tego typu instalacji, została ona opisana szczegółowo w dokumentacji dostępnej w oficjalnym serwisie internetowym MySQL. Jeżeli zaś zainstalowaliśmy pakiet PHPTriad, serwer MySQL jest już zainstalowany — znajdziemy go w katalogu *c:\apache\mysql*.

Uwaga

Nawet jeżeli rzeczywiście instalowaliśmy PHPTriad, to w przypadku, gdy korzystamy z systemu operacyjnego Windows 2000 lub Windows NT, będziemy musieli korzystać z programu *WinMySQLAdmin.exe*, znajdującego się w katalogu *MySQL/bin*. Jest to wygodny, prosty interfejs, pozwalający uruchamiać MySQL jako usługę systemową. Po uruchomieniu aplikacja ta działa dalej w tle, widoczna na pasku systemowym jako ikona w kształcie ulicznych świateł sygnalizacyjnych.

Skąd pobrać i jak zainstalować MySQL?

Otwórzmy stronę *www.mysql.com/downloads/mirrors.html* i wybierzmy jak najbliższy serwer. Przejdźmy do działu *downloads* (pliki do pobrania). Wybierzmy najnowszą spośród stabilnych wersji (będzie oznaczona jako *stable*) i pobierzmy ją.

Następnie uruchamiamy pobrany plik instalatora. Zainstaluje on wszystko domyślnie w katalogu *c:\mysql* — dla naszych celów ta lokalizacja jest równie dobra jak każda inna. Jeżeli zmienimy katalog, miejmy na uwadze to, że albo będziemy musieli za każdym razem wędrować do nowego katalogu w trybie MS-DOS, albo utworzymy sobie skrót, który będzie uruchamiał tryb MS-DOS bezpośrednio w tym katalogu.

Istnieje kilka różnych graficznych interfejsów użytkownika dla MySQL. Dobrym wyborem jest stary, wypróbowany skrypt PHP, w rodzaju aplikacji PHPMyAdmin. Na lokalnym serwerze działać będzie bardzo szybko. Postrzegana prędkość jego działania będzie zbliżona do prędkości pliku wykonywalnego, obsługującego całą bazę, ponieważ zarządzając ją poprzez PHPMyAdmin, nie wykroczymy ani razu poza nasz własny komputer. Dodatkowa korzyść polega na tym, że tej samej aplikacji możemy używać do zarządzania naszymi bazami danych na docelowym, zdalnym serwerze WWW. Ponieważ postawiliśmy sobie za cel jak najwierniejsze kopiowanie działania serwera docelowego, to rozwiązanie wydaje się być najlepsze.

Dwa skompilowane, zawarte w plikach wykonywalnych interfejsy GUI proponowane są przez oficjalny serwis MySQL pod adresem *www.mysql.com*. Oba oparte są na podobnej koncepcji. Co ważne, przypominają wyglądem i działaniem interfejsy standardowych, lokalnych aplikacji bazodanowych dla Windows, jakie z pewnością mieliśmy już okazję używać. Jeżeli absolutnie nie radzimy sobie z pracą w wierszu poleceń trybu MS-DOS, wygodniej będzie nam zapewne użyć którejś z nich.

Uwaga

Nie liczmy na to, że skompilowany, działający na naszym lokalnym pececie interfejs do obsługi baz MySQL pozwoli nam również administrować bazą danych na serwerze zdalnym. Większość administratorów tak konfiguruje swoje serwery MySQL, aby nie można było łączyć się z nimi z komputerów poza ich własną domeną. Owszem, możemy z łatwością przesłać na zdalny serwer

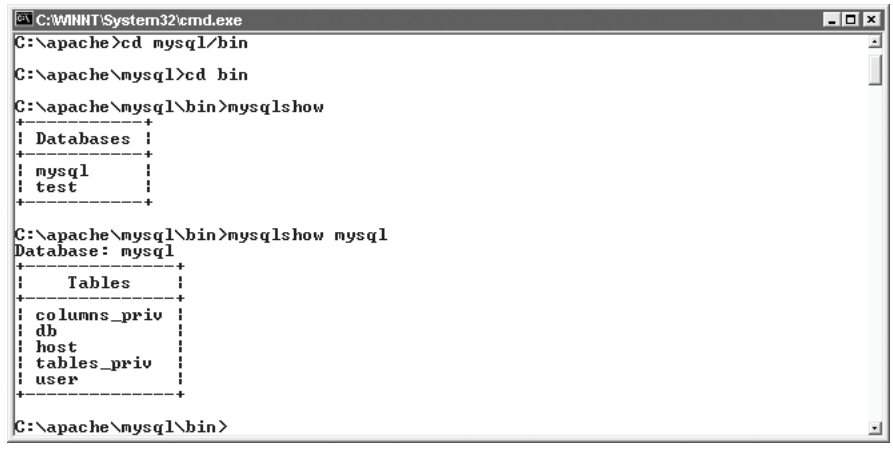
standardowy plik SQL i aktualizować bazę na serwerze docelowym właśnie w ten sposób. Nie można jednak zdalnie pracować na działającej na serwerze bazie danych, posługując się interfejsem uruchomionym na naszej lokalnej stacji roboczej. Użycie klienta zbudowanego w PHP ma tę zaletę, że przyzwyczaja nas do jednolitego wyglądu interfejsu i korzystania ze wspólnych konwencji dotyczących importu i eksportu plików między obydwoma środowiskami.

Mysqlshow

Aby obejrzeć bazy danych i tabele utworzone domyślnie w trakcie instalacji MySQL, uruchomimy załączoną do pakietu aplikację mysqlshow. Otwórzmy okno trybu MS-DOS (znajdziemy je przeważnie gdzieś w skrótach menu *Start/Programy/Akcesoria*) i przejdźmy do katalogu zawierającego pliki wykonywalne serwera MySQL — zwykle będzie to katalog *c:\mysql\bin*. Zawartość tego katalogu stanowić będą przede wszystkim pliki EXE. Najważniejsze, z których będziemy korzystać, to *mysqld* (serwer), *mysql* (klient), *mysqladmin* i *mysqlshow*. *Mysqlshow*, jak sama nazwa wskazuje, pokazuje (ang. *show*) nam w wierszu poleceń zawartość wskazanej bazy danych lub tabeli. Na dobry początek, sprawdźmy działanie poniższych poleceń:

```
mysqlshow mysql
mysqlshow mysql user
mysqlshow mysql user user
```

Pierwszy przykład da następujący rezultat:



Rysunek 12.2. *Rezultat użycia polecenia mysqlshow*

Znaczenie tych poleceń może nie być jasne, dopóki nie zobaczymy bazy danych o bardziej przejrzystym nazewnictwie, jednak bezpośrednio po instalacji dane znajdują się tylko w tej jednej bazie. Pierwszy parametr polecenia to nazwa bazy, następny — nazwa tabeli, a ostatni — nazwa konkretnego pola w tej tabeli. Gdy polecenie `mysqlshow` otrzymuje od użytkownika

jeden parametr (`mysql`), wyświetla nazwy tabel tworzących bazę danych o tej nazwie. Podając dwa parametry, otrzymujemy listę pól w tabeli wskazanej w drugim argumencie, razem ze szczegółowymi informacjami o typach kolumn. Podając wszystkie trzy, otrzymujemy tylko wskazane pole oraz informację, jaki to typ kolumny.

Licencja

Wygląda na to, że twórcy i opiekunowie serwera MySQL poszli na całość i oferują obecnie swój produkt całkowicie bezpłatnie w ramach licencji GNU GPL. Oznacza to, że możemy pobierać, instalować i korzystać z serwera MySQL bez żadnych ograniczeń, o ile tylko nie będziemy próbowali go sprzedawać. Pełne, aktualne i szczegółowe informacje na ten temat znajdziemy pod adresem: *www.mysql.com/products/index.html*. Kiedyś zakup licencji MySQL dla Windows kosztował około sto dolarów, na szczęście niektóre dobre rzeczy z czasem stają się jeszcze lepsze…

Przyspieszony kurs SQL

Język SQL — *Structured Query Language* — to wspólna mowa baz danych. Praktycznie wszystko, co będziemy robić z serwerem MySQL, wyrażone będzie w języku SQL. Bieżąca część rozdziału ma za zadanie dostarczyć nam wystarczająco dużo informacji, abyśmy mogli stworzyć i uruchomić kilka prostych, lecz praktycznych aplikacji MySQL, korzystających z PHP jako skryptowej części obliczeniowej oraz prezentacji Flash jako interfejsu użytkownika.

Pierwsze kroki

Już na samym początku potrzebne nam będą dane, na których możemy ćwiczyć. W przykładach znajdziemy plik o nazwie *practice.sql*. Zawartymi w nim danymi wypełnimy pierwszą pełną tabelę w nowej, ćwiczebnej bazie danych.

Zacznijmy od skopiowania pliku *practice.sql* do katalogu, w którym znajdują się pliki wykonywalne serwera baz danych MySQL (będzie to najprawdopodobniej *c:\mysql\bin* lub *c:\apache\mysql\bin*). Następnie otwieramy okno trybu MS-DOS i ustawiamy w nim ten katalog jako katalog bieżący.

Gdy już tam jesteśmy, sprawdzamy, czy aby na pewno działa program *mysqld.exe*, po czym wpisujemy następujące polecenia:

```
mysqladmin create practice
mysql -u root practice < practice.sql
```

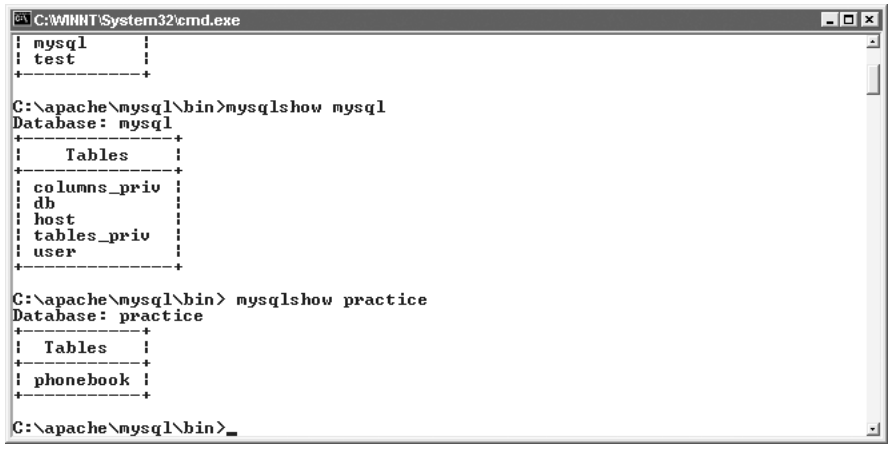
Pierwszy wiersz tworzy bazę danych o nazwie *practice* — ściślej, tworzy ją program mysqladmin. Jest to aplikacja używana przede wszystkim przez administratora serwera baz danych, nie musimy więc zawracać sobie głowy nauką dostępnych dla tego programu poleceń. Drugi wiersz zawiera bardzo często przydające się polecenie, powinniśmy więc postarać się zapamiętać jego składnię.

To polecenie każde serwerowi MySQL wykonać wszystkie instrukcje zawarte w pliku *practice.sql* dokładnie tak samo, jak gdybyśmy każdą z nich po kolei wprowadzali z wiersza poleceń. Jest to konieczność w przypadku, gdy mamy do czynienia z bardziej skomplikowanymi bazami danych lub z większymi ilościami danych, gdyż wprowadzanie wszystkiego z poziomu wiersza poleceń byłoby wówczas zbyt kłopotliwe. Plik sformatowany podobnie jak *practice.sql* możemy wyeksportować z niektórych programów — ten plik został wygenerowany za pomocą zbudowanego w PHP interfejsu PHPMyAdmin.

Za pomocą polecenia mysqlshow sprawdzamy, czy dane znajdują się w bazie danych.

```
mysqlshow practice
```

Po wydaniu polecenia powinniśmy zobaczyć coś podobnego do rezultatów przedstawionych na rysunku 12.3 (przedstawia on zawartość bazy *practice* czyli tabelę *phonebook*).



Rysunek 12.3. *Zawartość bazy danych practice*

Logowanie do MySQL

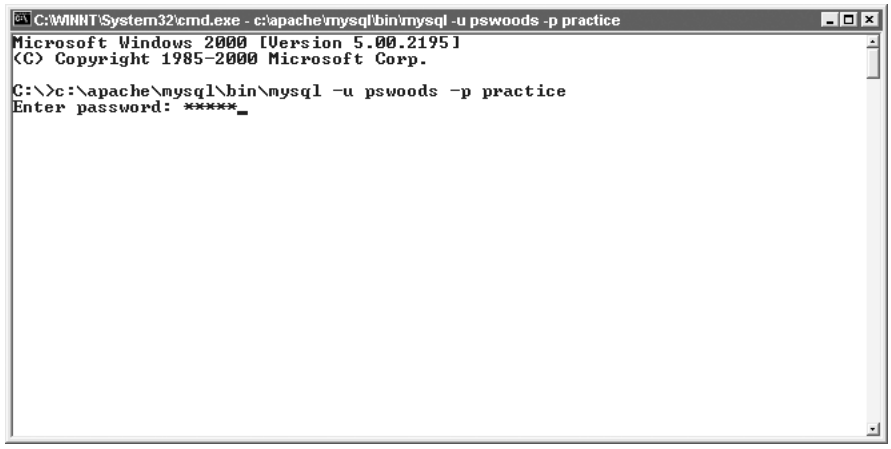
Skoro już utworzyliśmy i wypełniliśmy danymi tabelę *phonebook*, spróbujemy wydać kilka poleceń z poziomu wiersza poleceń MySQL. Poniższe polecenie uruchamia klienta mysql, w którym następnie sprawdzimy działanie kilku prostych zapytań SQL.

```
mysql -u root practice
```

Składnie tego polecenia wygląda następująco:

```
mysql -u nazwaUzytkownika (-p) (-h nazwaHosta) ( nazwaBazyDanych)
//przyklad: mysql -u pswoods -p -h 255.255.255.255 wwwPages
```

Zmienne ujęte w nawiasach nie są konieczne. Opcja `-p` każe serwerowi zapytać użytkownika o hasło, jak na rysunku 12.4. W tym rozdziale nie korzystamy jednak z haseł, przede wszystkim po to, aby zapewnić jak najwyższą zgodność i zmniejszyć liczbę zmian, które będziemy musieli ręcznie wprowadzić w konfiguracji MySQL, zanim przystąpimy do realizacji praktycznych przykładów aplikacji. W trakcie instalacji, MySQL konfiguruje się domyślnie dla jednego użytkownika — administratora (nazwa takiego użytkownika to *root*), bez żadnych haseł.



Rysunek 12.4. *Logowanie użytkownika do MySQL*

Parametr `-h` oraz następująca po nim nazwa hosta informują serwer MySQL, do którego hosta ma wysyłać zapytania SQL. Oto jedna z największych zalet MySQL — część serwerowa tej aplikacji może działać na naszym superszybkim, podkręconym serwerze linuksowym, na którym działa Apache, zaś za pomocą komponentu MySQL działającego po stronie klienta lub za pomocą interfejsu PHP znajdującego się na innym komputerze możemy wysyłać zapytania do serwera. Bardzo często takie rozwiązanie jest stosowane przez dostawców usług internetowych, którzy wydzielają dla potrzeb obsługi baz MySQL osobny serwer fizyczny i pozwalają mu wymieniać informacje z serwerami HTTP, na których znajdują się korzystające z baz danych serwisy WWW. W takim przypadku konieczne jest użycie opcji `-h` wraz z nazwą hosta obsługującego MySQL.

Parametr `nazwaBazyDanych` mówi serwerowi MySQL, której bazy danych ma użyć do realizowania dalszych zapytań. Jeżeli nie dodamy tego parametru, łącząc się po raz pierwszy z MySQL, będziemy musieli później — już w trakcie działania klienta MySQL — użyć polecenia o składni `use nazwaBazyDanych`.

Gdy skończymy eksperymenty w wierszu poleceń klienta MySQL, wprowadzamy polecenie `\q` i naciskamy klawisz *Enter*, zamykając program.

Uwaga

W przykładach zawartych w tym rozdziale zakładamy, że użytkownik to *root* (administrator) bazy danych i że w związku z tym przy logowaniu nie musimy podawać hasła. Szanse, że dokładnie tak samo będziemy mogli poczynać sobie z docelowym, zdalnym serwerem WWW, są znikome. Założenie, o którym mowa, ma na celu zachowanie prostoty. Dzięki temu wszystkie przykłady będą mogły działać na naszym lokalnym serwerze bez konieczności późniejszego zmieniania ich. Najlepiej jednak testować swoje skrypty lokalnie, ustawiając na lokalnym serwerze MySQL to samo hasło i nazwę użytkownika, jakie zostały nam przydzielone przez administratora na serwerze docelowym. Dzięki temu wystarczy, że przeniesiemy skrypt na zdalny serwer i od razu będzie on działać.

Najważniejsze polecenia

W niniejszym rozdziale zaledwie dotykamy problematyki MySQL, zaś sam język SQL to kolejny, osobny temat-rzeka. Poniższe podsumowanie zostało pomyślane przede wszystkim jako podręczna ściąga z poleceń SQL, które będą przewijać się w dalszej części rozdziału, oraz kilku innych, które przydadzą się nam na co dzień.

CREATE

Wygodną cechą języka SQL jest zrozumiała postać poleceń — podobnie jak w przypadku JavaScript lub ActionScript, działanie polecenia SQL zwykle jest zgodne z jego nazwą. Polecenie `CREATE` (dosł. utwórz) w zakresie, w jakim będziemy go używać, jest funkcją tworzącą tabele. Poniższy przykład to polecenie SQL tworzące tabelę w naszej przykładowej bazie danych:

```
CREATE TABLE specials (
   pkey varchar(10) NOT NULL,
   name varchar(50) NOT NULL,
   category varchar(50) NOT NULL,
   price float DEFAULT '0' NOT NULL,
   description text NOT NULL,
  PRIMARY KEY (pkey)
);
```

Powyższe wyrażenie utworzyło tabelę o nazwie `specials`. Składa się ona z pól: `pkey`, `name`, `description` i `price`. Pole `pkey` oznaczone jest jako *primary key*, czyli klucz główny (kolumna, według której indeksowana jest tabela). Jeżeli znamy inne rodzaje baz danych lub po prostu mamy ogólne pojęcie o bazach danych, wiemy, że to właśnie klucz główny decyduje o prędkości i możliwościach bazy danych. Zasadniczo w MySQL nie ma konieczności ręcznego tworzenia klucza głównego — jeżeli nie wskażemy kolumny mającej służyć za klucz podstawowy, zostanie ona utworzona dla nas. W naszym przykładzie klucz główny przyjmuje wartości tworzone przez prezentację Flash, aby zilustrować określony przepływ danych, typowy dla internetowych aplikacji utworzonych w technologii Flash.

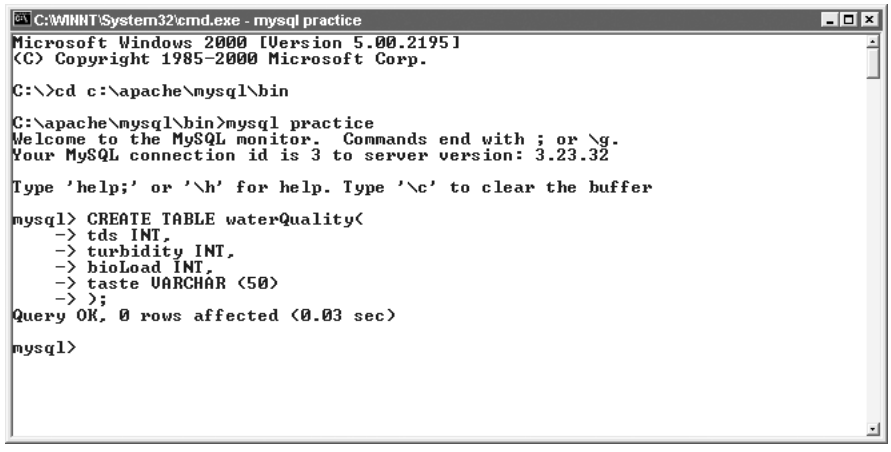
Podstawowa składnia polecenia CREATE jest następująca:

```
CREATE TABLE nazwatabeli(
nazwaKolumny TYPKOLUMNY ( dlugosc) [ opcje],
nazwaKolumny TYPKOLUMNY ( dlugosc) [ opcje],
...itd...
OGOLNE FUNKCJE TABELI
);
```

W przypadku prostych aplikacji omawianych w tym rozdziale, w każdej kolumnie każdej tabeli jako typu kolumny moglibyśmy użyć typu VARCHAR. Może on pomieścić od 1 do 255 znaków, albo też tyle, ile określimy. Łańcuch dłuższy od dopuszczalnego zostanie po prostu obcięty po ostatnim mieszczącym się znaku. Na przykład, jeżeli utworzymy kolumnę typu VARCHAR i nadamy jej długość 11, a następnie spróbujemy w nią wprowadzić tekst „wlazł kotek na płotek", w bazie danych pozostanie zapisany tylko łańcuch „wlazł kotek" — pierwsze 11 znaków.

Innym powszechnie spotykanym typem kolumny, którego będziemy wielokrotnie używać w tym rozdziale, jest typ INT. W tak zdefiniowanej kolumnie możemy zapisywać liczby całkowite od 0 do 4294967295, lub też o takiej liczbie cyfr, jaką określimy.

Spróbujmy dodać tabelę do bazy danych *practice*, definiując w niej kolumny typu VARCHAR i INT. Przestrzegajmy opisanej wyżej składni, pamiętając też o przecinkach na końcu opisu każdej kolumny i o średniku na końcu całego wyrażenia. Łatwo pogubić się przy wprowadzaniu długiego wyrażenia SQL, zawijającego się na końcu wiersza w oknie trybu MS-DOS w Windows (lub w wierszu poleceń Unix). Łatwym rozwiązaniem jest wprowadzanie w każdym wierszu tylko jednego elementu, jak w poprzednich przykładach. Nie ma obawy, klient MySQL nie przystąpi do interpretacji polecenia, dopóki nie wprowadzimy kończącego je średnika i nie wciśniemy *Enter*, jak widać na rysunku 12.5:



Rysunek 12.5. *Polecenie SQL wprowadzone w wierszu poleceń klienta MySQL*

Wskazówka

Składnia MySQL nie wymaga używania wielkich liter w pisowni nazw predefiniowanych obiektów SQL, jest to jednak godne uwagi zalecenie. Używając wielkich liter w poleceniach SQL i małych liter w nazwach kolumn i innych zmiennych, znacznie poprawiamy czytelność zapytań, ponieważ taka konwencja ułatwia szybkie rozróżnianie funkcji i zmiennych, na których te funkcje działają. Ponadto taka pisownia zapewnia naszej bazie danych zgodność ze wszystkimi bazami danych korzystającymi z SQL.

Dotychczas omówiliśmy jedynie dwa typy kolumn. Aby uprościć i bardziej ukierunkować nasze rozważania, będą to jedyne typy, jakie napotkamy, budując w tym rozdziale aplikacje Flash współpracujące z MySQL. Tabela 12.1 zawiera kilka innych powszechnie spotykanych i przydatnych typów kolumn, które być może okażą się nam niebawem przydatne.

Tabela 12.1. *Przydatne typy kolumn*

Typ kolumny	Domyślny zakres	Opis
`BIGINT`	od 0 do 18446744073709551615	Bardzo duża liczba całkowita
`FLOAT[(M,D)]` np.: `price FLOAT (5,2)`	od –3.402823466E+38 do 3.402823466E+38	Liczba zmiennoprzecinkowa, gdzie M to całkowita liczba cyfr, a D to liczba miejsc po przecinku
`DATE`	od 1000-01-01 do 9999-12-31	
`TIMESTAMP[(M)]` np.: `tstamp TIMESTAMP (12)`	od 1970-01-01 00:00:00 do roku 2037	Wyświetla wartości `TIMESTAMP` w formacie YYYYMMDDHHMMSS, jeżeli M wynosi 14; w formacie YYMMDDHHMMSS, jeżeli M wynosi 12; w formacie YYYYMMDD, jeżeli M wynosi 8, lub w formacie YYMMDD, jeżeli M wynosi 6. Za pomocą tego typu pola można porządkować wpisy w księdze gości lub na forum dyskusyjnym
`TEXT`	od 0 do 65535	Ogromny łańcuch znaków
`LONGTEXT`	od 0 do 16777215	Niewyobrażalnie wielki łańcuch znaków

Umiejętność tworzenia tabel jest przydatna zwłaszcza wtedy, gdy chcemy tworzyć je dynamicznie, za pomocą działającego po stronie serwera skryptu. Z drugiej strony, coraz więcej jest graficznych interfejsów dla baz MySQL. W tworzeniu tabel równie pomocne okazują się też interfejsy opracowane w PHP, np. wspomniany już PHPMyAdmin.

Bez względu na to, czy zechcemy zapamiętać składnię polecenia CREATE, czy też nie, powinniśmy mimo wszystko poświęcić trochę czasu na opanowanie pozostałych poleceń języka SQL, ponieważ będziemy korzystać z większości spośród nich przy każdej dynamicznej aplikacji bazodanowej, jaką przyjdzie nam tworzyć.

SELECT

SELECT (dosł. wybierz) to funkcja używana w dalszej części tego rozdziału najczęściej, a być może w ogóle jest najczęściej używana. Służy ona do wydobywania danych z bazy na podstawie określonych przez nas parametrów. Poniższe wyrażenia SQL zwracają wszystkie pola z każdego rekordu tabeli practice.phonebook (taka składnia — nazwaBazy.nazwaTabeli określa tabelę we wskazanej bazie; ponieważ w książce korzystamy wyłącznie z bazy *practice*, więc w przykładowych poleceniach składnia ta nie jest wykorzystywana — *przyp. red.*) (pierwsze wyrażenie) oraz tylko pola lastname i firstname (drugie wyrażenie) z tej samej tabeli.

```
SELECT * FROM phonebook ;
SELECT lastname, firstname FROM phonebook;
```

WHERE (dosł. gdzie) to parametr używany w połączeniu z poleceniem SELECT i wieloma innymi funkcjami SQL. Modyfikuje on wyrażenie, ograniczając je do następujących po nim warunków. W poniższym przykładzie zapytanie zwraca wszystkie pola z każdego rekordu, w którym pole lastname zawiera wartość monk:

```
SELECT * FROM phonebook WHERE lastname='monk' ;
```

LIKE (dosł. jak; czytaj: takie, że) umożliwia stosowanie znaków wieloznacznych, jak w poniższym przykładzie. Znak % (procent) zastępuje w szablonie dowolną liczbę jakichkolwiek znaków. Przytoczone tu zapytanie SQL zwróci wszystkie pola rekordów z tabeli practice.phonebook, dla których wartość zapisana w kolumnie lastname rozpoczyna się od litery m:

```
SELECT * FROM phonebook WHERE lastname LIKE 'm%' ;
```

ORDER BY (dosł. uporządkuj według) robi dokładnie to, czego można by się spodziewać na podstawie nazwy tego wyrażenia — porządkuje wyniki zapytania według wskazanej kolumny. Za pomocą dodatkowego parametru określamy kolejność uporządkowania: rosnącą (ASC) lub malejącą (DESC). Poniższy przykład zwraca listę nazwisk, imion i numerów telefonów, uporządkowaną rosnąco według kolumny *lastname*. Wyświetlane są tylko te rekordy, w których numer kierunkowy telefonu (pierwsze trzy cyfry) to 555.

```
SELECT firstname,lastname,phone FROM phonebook WHERE phone LIKE '555%'
   ORDER BY lastname ASC;
```

Tych kilka prostych odmian funkcji SELECT może oddać nam ogromne przysługi w zakresie operacji na danych. Na tym etapie potrafimy już wydobyć z bardzo prostej bazy danych interesujące nas informacje i uporządkować je wedle naszych upodobań.

INSERT

INSERT (dosł. wstaw) jest funkcją służącą do wprowadzania danych do bazy, po jednym wierszu. Pierwszy z poniższych przykładów przedstawia sposób informowania serwera MySQL, które pola w danym wierszu mają zostać zapisane (niektóre pola możemy zechcieć pozostawić puste, lub też mogą one zostać zapisane już wcześniej, automatycznie). Inne, mniej skomplikowane rozwiązanie polega na podaniu serwerowi wartości, które chcemy wprowadzić do bazy, i zostaną one wówczas zapisane w kolejnych polach rekordu na podstawie istniejącej kolejności kolumn:

```
INSERT INTO phonebook (lastname,firstname,address,city,state,zip,email)
VALUES ('smith','john','2347 Fondue Circle','Squaresville' ,'CA','90210',
'jsmith@email.com') ;
//...lub po prostu:
INSERT INTO phonebook VALUES
('smith','john','2347 Fondue Circle','Squaresville' ,'CA','90210',
'jsmith@email.com') ;
```

UPDATE

UPDATE (dosł. zaktualizuj) działa podobnie do INSERT, z tym że zmienia wartości w kolumnach już istniejących rekordów. Poleceniu UPDATE muszą towarzyszyć pewne kryteria, na podstawie których ma ono „wiedzieć", które rekordy ma zmodyfikować — zwykle występuje więc w połączeniu z WHERE, jak widać niżej:

```
UPDATE phonebook SET lastname='Jingleheimer-Schmidt', firstname='John'
   WHERE lastname='Woods' ;
```

Za pomocą polecenia UPDATE modyfikujemy przeważnie preferencje użytkownika, zmieniające się sumy itp. Jest to jedno z najczęściej używanych poleceń SQL; służy do zwykłej, codziennej obsługi i modyfikacji bazy danych.

DELETE

Polecenie DELETE (dosł. usuń), zgodnie z nazwą, usuwa rekord lub rekordy, jakie mu wskażemy. Poniższy przykład usuwa z tabeli practice.phonebook wszystkie rekordy, w których kolumna lastname ma wartość "jones".

```
DELETE FROM phonebook WHERE lastname = 'jones' ;
```

DROP

DROP (dosł. puść) to specjalne słowo kluczowe, stosowane wobec obiektów większych niż rekordy tabel — na przykład względem całych tabel. Należy używać tego polecenia ostrożnie. Poniższe wyrażenie SQL usunie z bazy danych tabelę o nazwie phonebook:

```
DROP TABLE phonebook;
```

Dostęp do bazy danych za pomocą PHP

Jeżeli nadal pamiętamy, czego nauczyliśmy się w poprzednim rozdziale, przyswojenie treści najbliższych kilku stron będzie bardzo łatwe. W momencie pojawienia się serwisów WWW z zapleczem bazodanowym język PHP jeszcze nie był w pełni ukształtowany, dlatego oferuje on obecnie rozbudowane i zarazem proste w użyciu mechanizmy obsługi baz danych.

Uwaga

Baza danych, której będziemy używać w poniższych przykładach, została zredukowana do najprostszej możliwej postaci, aby oszczędzić nam wdawania się w niepotrzebne szczegóły. Jeżeli planujemy zaprojektowanie bazy danych, z którą będziemy prowadzić wymianę danych, musimy się w tej dziedzinie dodatkowo dokształcić. Projektowanie baz danych to temat bardzo szeroki, pełen wielu subtelnych aspektów i wymagający zaawansowanej wiedzy. Gdybyśmy stwierdzili, że po przeczytaniu tego rozdziału posiedliśmy wystarczające podstawy, aby brać sprawy w swoje ręce, co najmniej kilku wytrawnych specjalistów zajmujących się bazami danych na co dzień mogłoby się na nas obrazić.

Dane pobrane z bazy na stronie HTML

Aby zacząć omawiać pierwszy przykład, utwórzmy bazę danych o nazwie `PSWine`. Będzie nam także potrzebna wypełniona danym tabela o nazwie `specials`. Utworzymy ją i wypełnimy za pomocą pliku *specials.sql*, używając w tym celu przedstawionych niżej poleceń (uprzednio należy skopiować plik *specials.sql* do katalogu *mysql/bin*):

```
mysqladmin create PSWine
mysql -u root PSWine < specials.sql
```

Omawiany tu przykład, plik *specialsHTMLOnly.php*, wysyła proste zapytanie SQL do bazy danych *PSWine*, po czym wyświetla rezultaty w postaci tabelki sformatowanej za pomocą znaczników HTML. Jest to jedna z podstron serwisu WWW fikcyjnej winiarni PSWoods Vineyards — w tym przypadku wyświetla ona specjalności dnia. Niżej widzimy kod skryptu PHP łączącego się z bazą danych i wykonującego zapytanie:

```
<HTML>
<BODY>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("PSWine",$db);
$result = mysql_query("SELECT * FROM specials",$db);
```

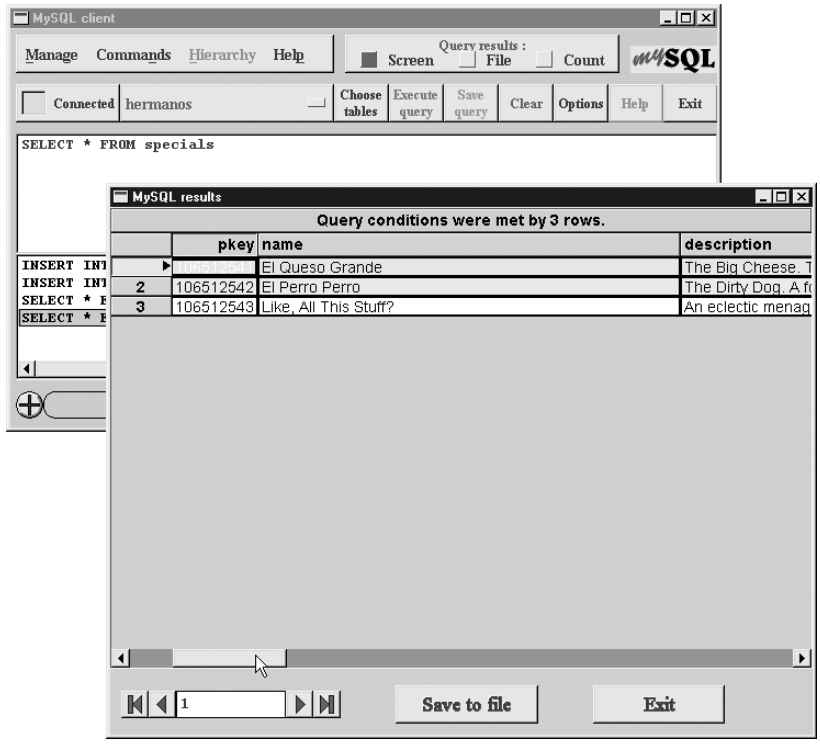
Pierwsza funkcja MySQL w tym skrypcie, `mysql_connect()`, nawiązuje połączenie z serwerem baz danych za pomocą takich samych parametrów, jakie wprowadzaliśmy ręcznie w wierszu poleceń klienta MySQL. W tym przykładzie `localhost` to domena, zaś `root` to nazwa użyt-

kownika. Gdyby w tym poleceniu miał znaleźć się także argument mieszczący hasło, należałoby go podać bezpośrednio po nazwie użytkownika. Tworzymy tym sposobem obiekt nazwany tutaj `$db` (choć można było nazwać go całkiem dowolnie), którym będziemy się posługiwać w dalszej części skryptu, gdy tylko zajdzie potrzeba połączenia się z bazą danych.

Następnie funkcja `mysql_select_db()` wybiera bazę danych *PSWine* jako bieżącą. Jak widać, drugim argumentem tej funkcji jest obiekt `$db`. W ostatni wierszu, zmiennej `$result` przypisywany jest za pomocą funkcji `mysql_query()` specjalny typ obiektu, którego pierwszym argumentem jest dowolne poprawne wyrażenie SQL, drugim zaś — nazwa obiektu połączenia, czyli `$db`. Jeżeli zapytanie, jakiego musimy tu użyć, jest wyjątkowo długie i złożone, lepiej zapisać je osobno, w postaci zmiennej łańcuchowej:

```
$sql="SELECT * FROM specials";
$result = mysql_query($sql, $db);
```

W ten sposób udało nam się umieścić całe wyrażenie SQL w osobnym wierszu, co ułatwi jego odczytanie i — w razie potrzeby — odnalezienie błędu. Łatwiej też skopiować zapytanie do interfejsu użytkownika MySQL i sprawdzić, czy na pewno nie ma błędu w jego składni, jak widać na rysunku 12.6:



Rysunek 12.6. *Sprawdzamy poprawność zapytania SQL w graficznym interfejsie klienta MySQL*

W tym miejscu w skrypcie pojawia się obiekt `$result`, zawierający wynik zapytania MySQL. Nie jest on jeszcze sformatowany tak, aby można było od razu wyświetlić jego zawartość. Potrzebna jest jeszcze jedna operacja. Obiekt `$result` jest czymś w rodzaju bezkształtnego pojemnika, mieszczącego nieokreśloną liczbę wierszy i kolumn, zależnie od tego, jakie dane z odpowiedzi na zapytanie zwróciła baza danych. Musimy przetworzyć zawartość obiektu `$result` i wydobyć z niego sensownie wyglądające dane. Istnieje szereg metod przeprowadzenia tego zabiegu, niżej przedstawiamy jedno z najbardziej poręcznych narzędzi, jakiego można tu użyć:

```
echo "<table cellpadding=10> ";
echo
"<tr><td><b>Specials</b></td><td><b>Description</b></td><td><b>Price</b></td></tr>\n";
while ($myrow = mysql_fetch_array($result)) {
   printf("<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>",
      $myrow["name"],$myrow["category"],$myrow["description"],
      $myrow["price"]);
   }

echo "</table>\n";
?>
</BODY>
</HTML>
```

Funkcja `mysql_fetch_array()` przetwarza w pętli po kolei każdy wiersz danych zawarty w zmiennej `$result` i tworzy dla każdego wiersza osobną tablicę asocjacyjną, w której kluczami są nazwy kolumn. Przy każdym przebiegu pętli skrypt wyświetla kolejny wiersz sformatowanej za pomocą znaczników HTML tabeli, po kolei wkładając w jego komórki poszczególne elementy takiej tablicy.

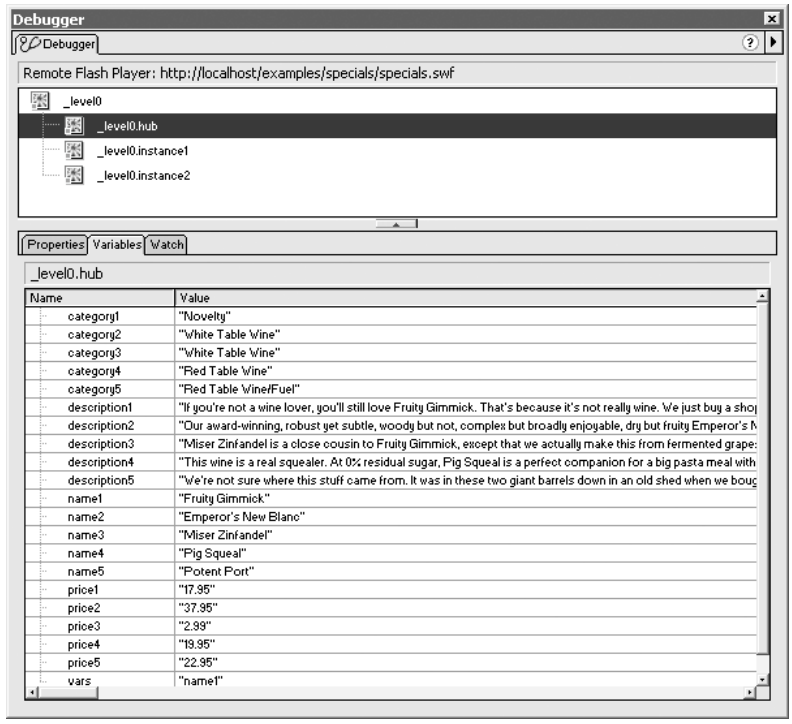
Rezultat końcowy to przejrzysta, elegancko wyświetlona zawartość tabeli `specials`. Aby ją odczytać z bazy i wyświetlić na stronie HTML, wystarczyło niecałe 10 wierszy skryptu. Oczywiście, dobrze byłoby dodać tu jeszcze wyrażenie warunkowe na wypadek, gdyby funkcja `mysql_fetch_array()` nie mogła zostać wykonana, takie rozwiązanie zostało tu jednak pominięte dla zachowania przejrzystości.

Dane pobrane z bazy w aplikacji Flash

Osiągniemy teraz ten sam efekt, tyle że dane nie zostaną wyświetlone na stronie HTML, lecz w prezentacji Flash. Plik nazywa się *specials.fla*, zaś poniższy kod pochodzi z pierwszej klatki jego głównej listwy czasowej. Działanie tej aplikacji można obejrzeć, otwierając plik *specials.html*.

```
loadVariables ("specialsSWF.php", "hub", "GET");
```

Odwołanie do pliku *specialsSWF.php* odbywa się metodą *GET*. Dane zwrócone przez skrypt zostaną umieszczone w klonie pustego klipu filmowego o nazwie `hub`. Możemy przekonać się, że klip `hub` rzeczywiście zostaje napełniony danymi, posługując się debuggerem, jak na rysunku 12.7.



Rysunek 12.7. *Zawartość klipu filmowego hub, odczytana z bazy danych*

Poniższy kod pochodzi z pliku *specialsSWF.php* i niewiele różni się od poprzedniego przykładu, tyle że tym razem dane końcowe sformatowane są specjalnie dla prezentacji Flash:

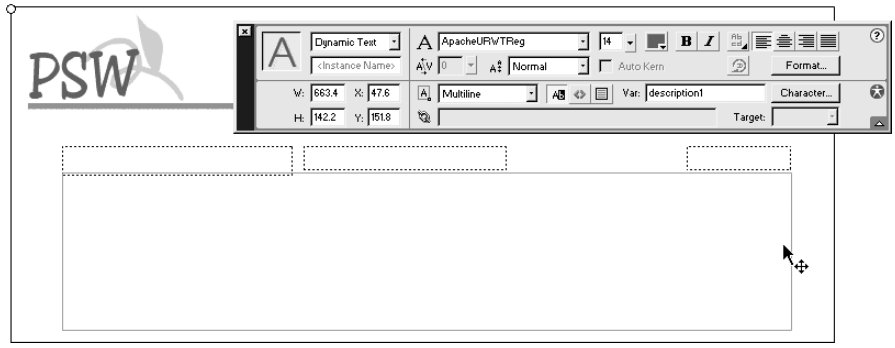
```
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("PSWine",$db);
$result = mysql_query("SELECT * FROM specials",$db);

$counter=1;
while ($myrow = mysql_fetch_array($result)) {
   printf("&name$counter=%s&description$counter=%s&price$counter=%s",
   rawurlencode($myrow["name"]),rawurlencode($myrow["description"]),
   rawurlencode($myrow["price"]));
   $counter++;
}
?>
```

HTML nie jest tu potrzebny, ponieważ ta strona nigdy nie będzie bezpośrednio budowana na podstawie kodu przez przeglądarkę. Skrypt wywoływany jest prosto z odtwarzacza Flash Player, a uzyskane wyniki interpretowane są już wewnątrz prezentacji Flash, w kontekście akcji `loadVariables()`. Funkcja `rawurlencode()` ujmuje dane wyjściowe w postaci zakodowanego łańcucha URL, który ma większe szanse dotarcia do prezentacji Flash w stanie nienaruszonym.

Przedstawiony niżej fragment kodu pochodzi z klonu klipu filmowego `hub`, umieszczonego w lewym górnym narożniku sceny. Wykonywany jest, gdy prezentacja otrzymuje z powrotem dane ze skryptu PHP.

```
onClipEvent (data) {
   for (vars in this) {
     _root[vars] = this[vars];
   }
}
```

Gdy skrypt odsyła zmienne z powrotem do prezentacji Flash w jej ulubionym formacie — `onClipEvent(data)` — pętla `for...in` przetwarza po kolei nadesłane dane i kopiuje je do poziomu `_root` prezentacji. Takie rozwiązanie działa, ponieważ pola z dynamicznym tekstem, w których wyświetlane są pobrane z bazy dane, umieszczone są właśnie na głównej listwie czasowej, a więc na poziomie `_root`, jak widać na rysunku 12.8. To już cały kod zawarty w tej aplikacji. Bardzo proste.



Rysunek 12.8. *Zwrócone przez skrypt PHP dane umieszczane są na scenie w dynamicznych polach tekstowych*

Statyczna struktura pól tekstowych na scenie wypływa z chęci zachowania prostoty i przejrzystości wykładu. Założyliśmy tu, że zawsze będzie dokładnie pięć win oferowanych jako specjalności dnia — to dość sztywne założenie jak na dynamiczną aplikację WWW. Z drugiej strony, gdy zrozumiemy zasadę działania tego przykładu, nie trzeba będzie nawet specjalnie ruszać głową — wystarczy trochę kodowania, projektowania — i z pewnością wymyślimy bardziej realistyczne rozwiązanie.

Modyfikowanie rekordów — moduł administracyjny

Zanim weźmiemy pod lupę kod następnego przykładu, sprawdźmy, jak działa on na lokalnym serwerze WWW. Przetestujmy go kilka razy, aż nabierzemy pewności, że wiemy, do czego służy. Przykład ten to *specialsAdmin.php*. Zadaniem tej aplikacji jest umożliwienie właścicielowi

naszego modelowego serwisu — zapewne samemu Panu PSWine — aktualizować rekordy w bazie danych `specials`, i to bez konieczności posiadania jakiejkolwiek wiedzy technicznej.

Uwaga

Podobnie jak w przypadku omawianego w poprzednim rozdziale przykładu *recursiveForm.php*, poniższy skrypt wygeneruje błąd PHP, jeżeli nie wyłączyliśmy na naszym lokalnym serwerze raportowania błędów niekrytycznych — docelowy serwer, dla którego piszemy nasze aplikacje, z pewnością będzie miał tę opcję wyłączoną. Także i tym razem, jeżeli z jakichś względów nie chcemy jej wyłączać, po prostu dodajmy w tym skrypcie zapis inicjujący zmienną (`$edit=""`). Pociągnie to za sobą także konieczność modyfikacji warunku na `if($edit="")` lub inny poprawny zapis o tym samym znaczeniu.

Kliknijmy łącze *Edit* w dowolnym wierszu aplikacji i wprowadźmy w polu tekstowym formularza HTML dowolny tekst. Gdy uznamy zmiany za zakończone, prześlijmy formularz (przycisk *Submit*) i zauważmy rezultat wprowadzonych zmian. Skrypt nie poprzestaje na wyświetlaniu zmiennych z tablicy `$HTTP_POST_VARS`; dane zmieniane są naprawdę w samej bazie danych. Można się o tym przekonać — zamknijmy przeglądarkę, otwórzmy ją ponownie i powróćmy do skryptu. Wartości nadal pozostają zmienione.

Gdy otwieramy stronę wygenerowaną przez ten skrypt po raz pierwszy, jeszcze bez zmiennych zakodowanych w łańcuchu URL w pasku adresowym przeglądarki, dotrzemy do poniższego fragmentu wyrażenia warunkowego. Jego działanie sprowadza się właściwie do wyświetlenia tabeli z zawartością aktualnej oferty specjalnej (tabela `specials`), z łączem do edycji każdego rekordu.

```
//powrot do modulu - wywolanie skryptu bez zmiennych w lancuchu URL ani zapytaniu POST
}else{
   drawTable();
}
//rysujemy tabele z lista win specjalnych
function drawTable(){
    $db = mysql_connect("localhost", "root");
    mysql_select_db("PSWine",$db);
    $result = mysql_query("SELECT * FROM specials",$db);

    echo "<table cellpadding=10> ";
    echo "<tr><td><b>Specials</b></td><td><b>Category</b></td>
    <td><b>Description</b></td><td><b>Price</b></td></tr>
    <td></td>\n";

    while ($myrow = mysql_fetch_array($result)) {
    $id=$myrow["pkey"];
    printf("<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td>
       <td>%s</td></tr>", $myrow["name"],$myrow["category"],
       $myrow["description"],$myrow["price"],
       "<a href=\"$PHP_SELF?edit=$id\">edit</a>");
    }
    echo "</table>\n";
}
```

```
?>
</BODY>
</HTML>
```

Pierwsze, co napotykamy, to funkcja `drawTable()`. Jak widzimy, składnia funkcji jest w PHP taka sama, jak w JavaScript i ActionScript.

Funkcja ta zawiera prawie to samo, co w poprzednich przykładach, z wyjątkiem hiperłącza. Pole `pkey` pełni rolę klucza głównego tablicy `specials`. Kolumna ta zawiera w każdym wierszu unikatowy element, którego serwer baz danych używa do szybkiego sortowania rekordów. Odwołując się do rekordów poprzez podanie klucza głównego, mamy pewność, że otrzymamy dokładnie ten rekord, o który nam chodzi. Wartość zaczerpnięta z tego pola jest dołączana do łańcucha URL w zmiennej o nazwie `edit`.

Dalej następuje ta część skryptu, która pozwala edytować zawartość wybranego rekordu — zapisane w nim informacje wyświetlane są w tabelce HTML. Ponownie wkraczamy więc na znajome terytorium — tym razem jednak zapytanie SQL w funkcji `mysql_query()` odpowiada jednemu, konkretnemu rekordowi, który chcemy edytować.

```
if($edit){
    $db = mysql_connect("localhost", "root");
    mysql_select_db("PSWine",$db);
    $result = mysql_query("SELECT * FROM specials WHERE
       pkey='$edit'",$db);
    $myrow = mysql_fetch_array($result);
?>
<!--opuszczamy tryb PHP, aby nie musiec recznie wstawiac znaku unikowego przed kazdym
➥z cudzyslowow -->
<form method="post" action="<?php echo $PHP_SELF ?>">
    <input type="text" name="name" value="<?php echo $myrow["name"] ?>">
    <input type="text" name="price" value="<?php echo $myrow["price"] ?>">
    <input type="text" name="category" value="<?php echo $myrow["category"] ?>">
    <br>
    <textarea name="description" cols="55" rows="15">
    <?php echo $myrow["description"]?></textarea>
    <br>
    <input type="hidden" name="id" value="<?php echo $myrow["pkey"] ?>">
    <input type="submit" name="update" value="update">
</form>
<?php
```

Na pierwszy rzut oka sporo kodu, ale pozory mylą — jest on w rzeczywistości bardzo prosty. Opuściliśmy pierwszy skrypt PHP i przeszliśmy na chwilę do kodu HTML. Nie opłaca nam się uciekać w tej sytuacji do dokumentu miejscowego (omawialiśmy go w poprzednim rozdziale), ponieważ symbole cudzysłowu w identyfikatorach tablicy, np. `$myrow["name"]`, są wymaganą częścią składni i muszą być interpretowane. Zamiast tego używamy więc kilku niewielkich fragmentów kodu PHP, wywołując interpreter i umieszczając aktualne wartości z poszczególnych kolumn rekordu w odpowiadających im polach formularza HTML. Zauważmy, że klucz główny modyfikowanego rekordu nadal przekazywany jest dalej — tym razem zaszyty w ukrytej zmiennej formularza, o nazwie `id`.

Na koniec, gdy użytkownik kliknie przycisk wysyłający formularz, poniższa część skryptu zapisuje wprowadzone przez formularz HTML dane, przesłane metodą *POST*, w bazie. Znajduje przy tym zastosowanie funkcja `UPDATE`, zawarta w zapytaniu SQL.

```
//administrator skonczyl wprowadzanie nowych danych dla konkretnego rekordu
}elseif($update){
    $db = mysql_connect("localhost", "root");
    mysql_select_db("PSWine",$db);
    $sql = "UPDATE specials SET name='$name', price='$price',
    description='$description' WHERE pkey='$id'";
    $result = mysql_query($sql, $db );
    drawTable();
```

Pomimo męczącego oko kodu HTML w środkowej części skryptu oraz niestrudzonego powtarzania kilku różnych funkcji `mysql_*`, przedstawiony skrypt nie należy do trudnych. Aby nadać kodowi bardziej elegancką formę, moglibyśmy wyodrębnić funkcje `mysql_*` w osobną funkcję, używającą zapytania SQL jako argumentu. Moglibyśmy też zawrzeć wstawiony w środek skryptu kod HTML w osobnym pliku i włączyć go w skrypt za pomocą polecenia `include`.

Tego typu aplikacja jest dla użytkownika bardzo cenna, ponieważ umożliwia mu proste zarządzanie treścią serwisu WWW. Jedno zagadnienie zawsze poruszane jest przy projektowaniu nowej witryny, działającej na podstawie bazy danych — w jaki sposób dane te będą wprowadzane i aktualizowane? Przeważnie sprawdza się następująca zasada: im mniej klient musi się uczyć, tym więcej jest skłonny zapłacić. I vice versa.

Podsumowanie — pocztówki sieciowe

Ostatnia aplikacja, jaką omówimy, bazuje na pojęciach wprowadzonych w tym oraz w poprzednim rozdziale. Wyposażeni w tę wiedzę, zbudujemy aplikację Flash wysyłającą elektroniczne pocztówki. Aplikacja ta składać się będzie z następujących elementów:

- Tabela *postcard* w bazie danych *PSWine*.
- Film Flash, w którym nadawca pocztówki wybierze zdjęcie i wprowadzi treść kartki. Ten plik to *postcard.fla*.
- Skrypt *postcard.php*, odpowiedzialny za zapisanie informacji wprowadzonych przez nadawcę w pliku *postcard.fla* do tabeli *postcard* w bazie danych.
- Film Flash użyty przez odbiorcę kartki do przeczytania wiadomości i obejrzenia dobranego do niej przez nadawcę zdjęcia. Ten plik nazwiemy *pickup.fla*.
- Skrypt *pickup.php* przejmuje wartość zmiennej pełniącej rolę klucza głównego (zmienna ta zawarta jest w łańcuchu URL) i przekazuje ją plikowi *pickup.swf*.
- Skrypt *retrieve.php*, który wprowadza dane do pliku *pickup.swf*.

Wszystkie potrzebne pliki należy, jak zwykle, skopiować do któregoś z katalogów wewnątrz katalogu *htdocs*. Następnie w tradycyjny sposób tworzymy nową tabelę w bazie danych *PSWine*.

```
mysql -u root PSWine < postcard.sql
```

Ostatnia rzecz, jaką musimy zrobić, aby przygotować aplikację do użycia, to dostosowanie ustawień SMTP w pliku konfiguracyjnym *PHP.ini*. Otwieramy go więc i odszukujemy w nim dwa wiersze, zbliżone wyglądem do przedstawionych niżej. Edytujemy je, podając w jednym z nich adres serwera SMTP, z którego korzystamy (jeżeli akurat go nie pamiętamy, odczytujemy go z ustawień klienta pocztowego), oraz nasz zwrotny adres e-mail. Oczywiście, dokonujemy tych zmian jedynie w przypadku, gdy naprawdę chcemy wysyłać wiadomość pocztą elektroniczną za pomocą naszej aplikacji.

```
SMTP = smtp-server ; for Win32 only
sendmail_from = pswoods@email.com ; for Win32 only
```

Dzięki temu zapisowi możemy korzystać w skryptach PHP z funkcji wysyłającej e-mailem pod wskazany adres powiadomienia o otrzymaniu pocztówki. Jeżeli zrezygnujemy z takiej możliwości lub jeżeli poprawne skonfigurowanie wysyłki powiadomień nastręcza nam kłopotów, możemy poprzestać na obejrzeniu gotowej kartki pod adresem *http://localhost/examples/postcard/pickup.php?id=982033627541th*, gdzie zamiast *postcard* należy wstawić nazwę katalogu wewnątrz katalogu *htdocs*, w którym zapisaliśmy ten przykład. Utworzona za pomocą pliku *postcard.sql* tabela *postcard* zawiera obecnie tylko jedną pozycję, powstałą przy jej wygenerowaniu.

Zaczynamy od końca — odbiór pocztówki

Zaczniemy od końca, śledząc bieg wydarzeń niejako z perspektywy adresata pocztówki, gdyż właśnie ta część aplikacji okaże się nam najbardziej znajoma. Oto kod źródłowy pliku *pickup.php*:

```
<HTML>
<BODY>
<?php
//jeżeli zmienna $submit posiada jakakolwiek wartosc (ktos wyslal formularz)...
if($id){
   foreach($HTTP_GET_VARS as $key => $value){
      $nameValPairs.=("$key=$value&");
   }
include ("SWFDisplay.inc");
//w przeciwnym wypadku wyswietl
}else{
    echo"it looks like this is not a valid pickup address.";
}
?>
</BODY>
</HTML>
```

Jest to przeróbka sztuczki poznanej jeszcze w rozdziale 9., gdzie dodawaliśmy zakodowane w łańcuchu URL pary nazw i wartości do kodu osadzającego na stronie prezentację Flash. Poniższy kod to niewielki fragment zagmatwanego pliku *SWFDisplay.inc* — zamieszczony tu dla przypomnienia:

```
...< PARAM NAME=movie VALUE="pickup.swf?<?php echo $nameValPairs ?>">...
...<EMBED src="pickup.swf?<?php echo $nameValPairs ?>"...
```

Występujące w nim małe fragmenty kodu PHP pobierają wartości ze zmiennej `$nameValPairs` i wstawiają je do filmu Flash. Zmienna `$nameValPairs` powstała w głównej części skryptu, w pętli `foreach...as`.

Następny krok to użycie przez aplikację *pickup.swf* informacji dostarczonych jej przez zaprezentowany przed chwilą skrypt PHP. Łańcuch URL będzie miał postać *http://host.com/postcard/pickpup.php?id=23094234098xx*. Jedyna zmienna brana pod uwagę to `id`, przechowująca wartość klucza głównego, identyfikującego rekord związany z daną kartką.

Poniższy fragment kodu pochodzi z pierwszej klatki głównej listwy czasowej w pliku *pickup.fla* i pokazuje, jak rozpoczyna się ów proces. Film Flash, z którego pochodzi ta akcja, jest bardzo podobny do znanego nam już *specials.fla*, w którym obiekt `hub` użyty był jako „pojemnik" na przysyłane dane.

```
loadVariables ("retrieve.php?id=" add id, "hub");
```

Gdy wywołany zostanie plik *retrieve.php*, któremu w łańcuchu URL zostanie dostarczona wartość zmiennej `id`, interpreter przetworzy poniższy kod. Ponownie mamy tu coś znajomego — tym razem podobny mechanizm występował w części aktualizującej dane w bazie w pliku *specialsAdmin.php*; różnica w tym przypadku polega na tym, że jedynie wyświetlamy pasujący rekord tabeli MySQL, nie zmieniamy zaś jego zawartości.

```
<?php
//jezeli zmienna $submit ma wartosc (uzytkownik wyslal formularz)...
if($id){
    $db = mysql_connect("localhost", "root");
    mysql_select_db("PSWine",$db);
    $result = mysql_query("SELECT * FROM postcard
       WHERE id='$id'",$db);

if ($myrow = mysql_fetch_array($result)) {
printf("%s=%s&%s=%s&%s=%s&%s=%s&",
"sender", rawurlencode($myrow["sender"]),
"recipient", rawurlencode($myrow["recipient"]),
"message", rawurlencode($myrow["message"]),
"mc", rawurlencode($myrow["mc"]));
   echo "errorMsg=here is your card.";
} else {
   echo "errorMsg=sorry - i couldn't find your card";
}
//nie podano $id. Wyswietl komunikat o bledach
}else{
   echo"errorMsg=sorry - your card is missing";
}
?>
```

Zauważmy, że wprowadziliśmy tu skromny mechanizm obsługi błędów. Jeżeli nie zostanie podany parametr `id` — lub jeśli MySQL nie znajdzie odpowiadającego danej wartości `id` rekordu w bazie danych — wówczas wyświetlany jest komunikat o błędzie. W przeciwnym razie skrypt zwraca dane wprowadzone przez nadawcę, wraz z komunikatem *Here is your card* (*oto pocztówka dla Ciebie*).

Gdy tylko dane zostają przekazane prezentacji Flash, jest wykonywany poniższy kod (pochodzi on z klipu filmowego `hub`, umieszczonego w lewym górnym narożniku sceny):

```
onClipEvent (data) {
    for (vars in this) {
       _root[vars] = this[vars];
    }
    loadMovie (mc + ".swf", "_root.ph");
   _root.ph._alpha=40;
}
```

To zasadniczo ten sam kod, który oglądaliśmy w pliku *specials.fla* — wzbogaciliśmy go jedynie o procedurę ładującą wskazany przez nadawcę zewnętrzny plik SWF. Podobnie jak w pliku *specials.fla*, dynamiczne pola tekstowe znajdują się bezpośrednio na scenie (na poziomie `_root`).

Wprowadzamy dane

Najpierw zobaczymy, w jaki sposób dane trafiają do tabeli *PSWine.postcard*. Kolejny klocek w tej układance, plik *postcard.php*, stanowi nową mieszankę znanych nam pojęć i mechanizmów (i starego kodu). Aplikacja Flash wprowadza do tego skryptu następujące zmienne: `sender`, `recipient`, `message` i `e-mail`:

```
<?php
//sprawdz poprawnosc zmiennej sender (musi zaczynac się dwoma znakami alfanumerycznymi
if (eregi("^[[:alnum:]]{2,}", $sender)) {

//sprawdz e-mail i wyslij wiadomosc
$emailPiece="[[:alnum:]]+";
if (eregi("^$emailPiece([-_.]?$emailPiece)*@$emailPiece([-_.]?$emailPiece)*$", $email)) {
```

Do tego miejsca nie zrobiliśmy jeszcze nic poza sprawdzeniem poprawności imienia nadawcy i adresu e-mail odbiorcy — wartości zmiennych wprowadzonych przez nadawcę. Powodem, dla którego chcieliśmy sprawdzić, czy imię nadawcy rozpoczyna się od dwóch znaków alfanumerycznych, jest system tworzenia wartości klucza podstawowego w tej tabeli — skrypt używa tych znaków jako jego części. Aby w razie potrzeby przypomnieć sobie działanie wyrażeń regularnych, możemy zajrzeć do poprzedniego rozdziału.

```
    $body="click here to pick up your card \n\n
    http://localhost/examples/postcard/pickup.php?id=$id";
    mail("$email","a postcard from $sender", "$body");
    echo "errorMsg=your postcard has been sent.";
}else{
  echo "errorMsg=invalid email";
}
```

```
//pomijamy tutaj, jezeli zmienna $sender nie zaczyna sie dwoma znakami alfanumerycznymi
}else{
echo "errorMsg=invalid sender name";
}
```

Nowością jest funkcja `mail()`. Wysyła ona wiadomość e-mail o treści określonej w ostatnim argumencie do odbiorców określonych w pierwszym argumencie, jako temat wiadomości przyjmując wartość argumentu środkowego. Tutaj używamy jej w składni `mail(adres, temat, tresc)`. Treścią wysyłanej wiadomości jest proste polecenie kliknięcia łącza do strony, którą już oglądaliśmy, *pickup.php*.

```
$db = mysql_connect("localhost", "root");
mysql_select_db("PSWine",$db);
$result = mysql_query("INSERT INTO postcard VALUES
('$id','$sender','$recipient','$email','$message','$mc')",$db);
?>
```

Na koniec, jeżeli użytkownik pomyślnie przejdzie testy przeprowadzane za pomocą wyrażeń regularnych, dane dodawane są do bazy danych za pomocą funkcji `INSERT` języka SQL. Rzecz ciekawa — przy wprowadzaniu wartości zmiennych w wyrażeniu SQL tego typu, nazwy zmiennych podajemy koniecznie w apostrofach.

Zanim damy nura w plik *postcard.fla*, czyli w interfejs użytkownika naszej aplikacji, przyjrzyjmy się kodowi formularza *genericForm.html* i wypróbujmy kilka razy jego działanie. Z tego właśnie pliku pochodzi poniższy kod HTML:

```
<form name="form1" method="post" action="postcard.php">
<input type="text" name="id" value="id">
<input type="text" name="mc" value="mc">
<input type="text" name="sender" value="sender">
<input type="text" name="recipient" value="recipient">
<textarea name="message">message</textarea>
<input type="text" name="email" value="email">
<input type="submit" name="Submit" value="Submit">
</form>
```

Przywołany przykład nie stanowi części omawianej tu aplikacji. Więcej — jest dziecinnie prosty. Dlaczego więc mamy przyglądać się w tym rozdziale tak prymitywnemu formularzowi, zwłaszcza że za chwilę mamy zagłębić się w naprawdę fascynujący materiał, coś, na co długo czekaliśmy — wykonany w programie Flash interfejs do wprowadzania danych do bazy MySQL? Ano dlatego, że choćby nie wiem jak skomplikowany był stworzony w programie Flash formularz, ma on przede wszystkim działać tak samo, jak prosty formularz HTML przedstawiony wyżej.

Otwieramy plik *postcard.fla*. Porozglądajmy się przez chwilę, zanim przystąpimy do analizy zawartego w nim kodu ActionScript. Przewodnią koncepcją, jak w każdej dobrej aplikacji, jest uszeregowanie obiektów, w tym klipów filmowych, zależnie od ich funkcji. Właśnie z tego względu, a także dlatego, że ta część aplikacji złożona jest z kilku różnych scen (wybór zdjęcia, wypełnienie pocztówki treścią, wysłanie danych i otrzymanie odpowiedzi), wszystkie elementy znajdujące się w bibliotece — poza obiektami `blank` i `hub` — pobierane są z biblioteki dynamicznie, w trakcie działania aplikacji.

Gdy rozpoczyna się odtwarzanie filmu, na scenie nie ma żadnego elementu graficznego. Większość zawartego w aplikacji Flash kodu odnosi się do przejść pomiędzy scenami oraz steruje poruszaniem się klipów filmowych w trakcie tych przejść.

Pierwszy fragment kodu, jakiemu się przyjrzymy, znajduje się w pierwszej klatce głównej listwy czasowej:

```
mcNames = new Array("dog", "sandwich");
loadMovie (mcNames[0]+".swf", "ph");
hub.mc = mcNames[0];
// wybieranie plikow SWF
attachMovie("pickImage", "pickImage", 1);
pickImage._x = 175;
pickImage._y = 50;
// dolacz nastepny MC
attachMovie("next", "next", 9);
next._x = 480;
next._y = 430;
```

Poza dość oczywistymi operacjami umieszczającymi na scenie klipy filmowe `pickImage` i `next`, w tej części skrypt inicjuje też zmienną tablicową `mcNames`, zawierającą listę zewnętrznych plików SWF służących jako graficzne tła pocztówek. Początkową wartość otrzymuje też zmienna `hub.mc`, na wypadek gdyby użytkownik zdecydował się nie zmieniać pierwotnego tła. Jeżeli chcielibyśmy zrobić coś naprawdę wyrafinowanego, moglibyśmy załadować tę tablicę dynamicznie ze skryptu PHP, który odczytałby zawartość katalogu przeznaczonego wyłącznie do przechowywania plików SWF zawierających wymienne tła.

Pierwszą czynnością użytkownika jest wybór graficznego tła. Służy do tego klip filmowy `pickImage`:

```
this.i=0;
```

Do przycisków zagnieżdżonych w klipie `pickImage` dołączony jest poniższy skrypt. Jest to kod ActionScript, odpowiedzialny za zamianę użytego w tle obrazka, gdy użytkownik klika przyciski w kształcie skierowanych w lewo i prawo strzałek.

```
//przycisk w lewo
on (release, releaseOutside) {
   i--;
   if (i== -1){
      i=_root.mcNames.length-1;
   }
   loadMovie (_root.mcNames[i] + ".swf", "_root.ph");
   _root.hub.mc=_root.mcNames[i];
}
//przycisk w prawo
on (release, releaseOutside) {
   i++;
   if (i==_root.mcNames.length){
     i=0;
   }
   loadMovie (_root.mcNames[i] + ".swf", "_root.ph");
   _root.hub.mc=_root.mcNames[i];
}
```

Kliknięcie dowolnego z tych przycisków powoduje wzrost lub spadek wartości zmiennej liczącej `i`, zawartej w klipie filmowym `pickImage`; jednocześnie do egzemplarza klipu `_root.ph` ładowany jest odpowiadający bieżącemu indeksowi `i` zewnętrzny plik SWF (gdyż, jak pamiętamy, listę zewnętrznych plików SWF zawiera tablica `_root.mcNames`).

W klipie filmowym `next` znajduje się następująca akcja:

```
on (release, releaseOutside) {
   _root.switchMode();
}
```

Uruchamia ona niestandardową funkcję `switchMode()`, umieszczoną na głównej listwie czasowej.

Funkcję tę prezentujemy niżej. Pozbywa się ona klipów filmowych `pickImage` i `next`, obsługujących system wyboru tła pocztówki w tym filmie. Znajdujący się w tle plik SWF zostaje rozjaśniony do tego stopnia, że nie koliduje wizualnie z nałożonym na niego formularzem (`_alpha=30`). Dołączany jest występujący na następnym etapie wysyłania kartki klip filmowy o nazwie `elFormo`:

```
//przejscie od wyboru obrazka do formularza
function switchMode () {
    next.removeMovieClip();
    pickImage.removeMovieClip();
    ph._alpha = 30;
    // dolaczamy klip filmowy elFormo
    attachMovie("elFormo", "elFormo", 9);
    elFormo._x = 125;
    elFormo._y = 70;
}
```

Poniższe akcje znajdziemy na listwie czasowej klipu *elFormo*:

```
function attachSend(){
this.attachMovie( "send", "send", 1 );
send._x=265;
send._y=280;
}
attachSend();
```

Szczególnie ważne jest, aby klip filmowy `Send` nie był umieszczony na scenie na stałe — w niektórych okolicznościach nie będziemy chcieli przecież dać użytkownikowi możliwości wysłania formularza poprzez kliknięcie zawartego w tym klipie przycisku. Ponieważ przycisk dołączany jest za pomocą funkcji `attachSend()`, można z tego faktu skorzystać i łatwo wywołać ją z dowolnego poziomu w filmie, gdy `elFormo` znajduje się na scenie.

Gdy użytkownik wypełnił już wszystkie pola formularza w `elFormo`, kolejnym logicznym posunięciem będzie kliknięcie przycisku zawartego w klipie `Send`. Ten właśnie przycisk uruchamia cały proces przetwarzania danych — jest do niego przypisana następująca akcja:

```
on (release, releaseOutside, keyPress "<Enter>") {
   _root.sender();
}
```

Pierwsze, co następuje w funkcji sender(), to znana nam już migracja danych od jednej ścieżki obiektu do drugiej.

```
// pojemnik na zmienne formularza
function sender () {
  elFormo.errorMsg = "adding your postcard to the database...";
  for (vars in elFormo) {
    // pozbywamy sie klonow bez nazw (przycisk "send")
      if(vars.indexOf("instance")==-1&&vars.indexOf("errorMsg")==-1){
        hub[vars] = elFormo[vars];
     }
}
```

Znaczenie wyrażenia warunkowego for...in w tym przypadku powinno być już oczywiste. Przenosimy zmienne zawarte w elFormo do klonu klipu filmowego o nazwie hub. Wyrażenie warunkowe wewnątrz pętli for...in odfiltrowuje klony nie posiadające nazw (przycisk) oraz wartość zmiennej errorMsg. Ponieważ nie musimy wysyłać tych zmiennych skryptowi PHP, nie ma potrzeby zapisywać ich w obiekcie hub.

```
// tworzymy unikatowe ID
elDato = new Date();
hub.id = elDato.getUTCTime()+hub.sender.substring( 0, 2)
elFormo.send.removeMovieClip();
hub.dataLoader();
}
```

Tym sposobem mamy już także unikatową wartość id dla aktualnie wysyłanej kartki; usuwamy przy tym przycisk Send z klipu elFormo i uruchamiamy funkcję dataLoader() wewnątrz klonu klipu filmowego hub. Zmienna id jest o tyle ciekawa, że zostanie zapisana w bazie danych jako klucz główny bieżącego rekordu, jednoznacznie odróżniający aktualnie wysyłaną pocztówkę od innych. Chociaż MySQL oferuje znacznie bardziej zaawansowane metody wyznaczania niepowtarzalnych wartości klucza głównego, to jednak utworzenie takiej wartości w aplikacji Flash oszczędza dodatkowej transmisji na serwer i skraca skrypt o pewną liczbę wierszy kodu. Wartość id zostaje przypisana za pomocą połączenia wartości zwróconej przez funkcję Date.getTime() — jest to dokładny czas w milisekundach, mierzony od daty 1 stycznia 1970 roku — oraz pierwszych dwóch znaków alfanumerycznych zaczerpniętych z imienia nadawcy.

Egzemplarz klipu filmowego hub, umieszczony w lewym górnym narożniku sceny, zawiera na swojej listwie czasowej kod funkcji dataLoader() — to nic innego, tylko proste wyrażenie z użyciem funkcji loadVariables, jakie spotykaliśmy we wszystkich poprzednich rozdziałach:

```
function dataLoader () {
   loadVariables ("postcard.php", this, "GET");
}
```

Oto zaś kod dołączony do klipu filmowego hub:

```
onClipEvent(data){
   _root.elFormo.errorMsg=this.errorMsg;
   if(this.errorMsg.indexOf("invalid") != -1){
      _root.elFormo.attachSend();
   }
}
```

W pierwszej kolejności, po otrzymaniu od skryptu wartości zmiennej `errorMsg` aplikacja powinna wyświetlić tę wartość za pośrednictwem klipu `elFormo` (czerwone litery w dolnej części sceny). Jeżeli następnie wyniknie problem z adresem e-mail lub imieniem nadawcy, do klipu `elFormo` ponownie dołączany jest klip z przyciskiem `Send` — i użytkownik może spróbować wysłać pocztówkę jeszcze raz.

Inna aplikacja, ten sam pomysł — księga gości

Gdy już poradzimy sobie z podstawowymi pojęciami i procedurami przesyłania danych tam i z powrotem za pomocą PHP pomiędzy serwerem MySQL i prezentacją Flash, możemy z łatwością zacząć przystosowywać posiadane zasoby kodu do użycia w nowych aplikacjach. Z punktu widzenia użytkownika będą one miały często zupełnie inne przeznaczenie i sposób działania, jednak proste zadanie przesyłania danych w obu kierunkach pozostanie bez zmian, gdy tylko odpowiednio dopracujemy je pod względem technicznym.

Dobrym przykładem różnej, a podobnie działającej aplikacji jest księga gości, dołączona do tego rozdziału w folderze *przyklady/guestbook*. Mimo że księga gości w oczywisty sposób różni się z punktu widzenia użytkownika od mechanizmu wysyłania elektronicznych pocztówek, znalazły w niej zastosowanie nie tylko niektóre fragmenty kodu PHP i ActionScript, ale także całe pliki FLA i kompletne skrypty PHP. Ogólnie rzecz biorąc, im bardziej udaje nam się uniezależniać główne funkcje od elementów charakterystycznych dla konkretnego rozwiązania, tym łatwiej później znaleźć dla takich funkcji nowe zastosowania.

Aby uruchomić księgę gości, najpierw tworzymy nową tabelę w bazie danych *PSWine*, posługując się w tym celu dołączonym plikiem *guestbook.sql*. Otrzymamy nową tabelę o kilku już wprowadzonych rekordach. Następnie otwieramy plik *guestbook.html*, aby obejrzeć rezultat. Osadzony w nim plik SWF zawiera łącze do osobnej strony, służącej do dodania nowego wpisu do księgi gości. Nie będziemy tu jednak analizować kodu, ponieważ znakomita jego większość pochodzi z poprzedniego przykładu. Warto też znaleźć chwilę i przejrzeć pliki FLA zapisane w folderze */przyklady/FLA*.