

# Bezpieczeństwo MySQL w pigułce

Przemek Sobstel (<http://sobstel.org>)

SegfaultLabs, sierpień 2006

W niniejszym artykule zostaną poruszone kwestie bezpieczeństwa systemu zarządzania bazą danych MySQL w zastosowaniach dla sieci Web (z naciskiem na PHP).

Zapewnienie bezpieczeństwa danych zgromadzonych w ramach całego systemu jest wynikiem wielu różnych działań. Tutaj nacisk położono na ochronę przed wstrzykiwaniem złośliwego kodu do zapytań SQL, zarządzanie uprawnieniami przez system bazodanowy oraz kwestie związane z zapewnieniem integralności danych.

## Ochrona przed iniekcją wrogiego kodu SQL

Iniekcja wrogiego kodu SQL (ang. *SQL Injection*) jest jednym z najdotkliwszych w skutkach ataków na internetowe aplikacje bazodanowe. Zagrożenie to jest związane bezpośrednio z danymi gromadzonymi w bazie danych, jednak nie wynika ono ze słabości systemu bazodanowego, lecz sposobu przetwarzania danych przez aplikację kliencką. Dlatego też zabezpieczenia powinny być głównie implementowane właśnie na poziomie aplikacji.

Ochrona przed wstrzykiwaniem kodu SQL ogranicza się do przestrzegania następujących zasad bezpieczeństwa [WWW1]:

- Weryfikowanie wszystkich danych wejściowych umieszczanych w zapytaniach SQL, m.in. sprawdzanie czy dane te są oczekiwanego typu. Wartości liczbowe mogą być także testowane pod kątem warunków granicznych [Shem05, s.40].
- Formatowanie danych nienumerycznych umieszczanych w zapytaniach SQL, specyficznymi dla danej bazy, funkcjami ucieczki.
- Unikanie połączeń z wykorzystaniem konta superużytkownika (posiadającego wszystkie prawa). Powinno się używać konta z najmniejszym wymagany zbiorem uprawnień.
- Użytkownikowi końcowemu nie powinny być wyświetlane żadne komunikaty błędów, zwłaszcza zawierające informacje na temat schematu bazy danych.

Solidne filtry poprawności danych wejściowych w pewnym stopniu podnoszą bezpieczeństwo (można w tym celu użyć np. biblioteki SafeSQL<sup>1</sup>), ale niestety nie są wystarczające [Shem05, s.40].

---

1 <http://www.phpinsider.com/php/code/SafeSQL/>

Rozwiązania oparte na wzorcach (sygnaturach), jako środek zapewniający wysoki poziom bezpieczeństwa, są mało efektywne, ponieważ [OfSh04, s.15; Shem05, s.40] :

- Baza sygnatur musi obejmować niemal wszystkie możliwe formy ataku. Taka baza jednak musiałaby być bardzo duża, co jest nie do zaakceptowania ze względu na wydajność. Ponadto, baza ta obejmowałaby tylko i wyłącznie znane formy ataku.
- Pewnym rozwiązaniem powyższego problemu jest stworzenie niewielu ogólnych sygnatur, np. zbiór ograniczający się do kilkunastu słów kluczowych i znaków specjalnych. W praktyce jednak, taki ograniczony zestaw sygnatur blokuje więcej normalnych użytkowników, niż włamywaczy.
- Kombinacja alternatywnego schematu kodowania (kodowanie URL, Unicode) i kreatywnego SQL omija większość filtrów bazujących na wzorcach.

Skuteczniejszą formą ochrony jest zastosowanie funkcji ucieczki, której zadaniem jest wstawianie znaków ucieczki przed znakami specjalnymi, dzięki czemu dane mogą być użyte w zapytaniach SQL. Najczęściej programiści PHP używają w tym celu funkcji `addslashes()` i `mysqli_real_escape_string()`. Jednakże ze względu na ataki wykorzystujące różne systemy kodowania znaków, bezpieczniejszym rozwiązaniem jest skorzystanie z zapytań preinterpretowanych<sup>2</sup> (ang. *prepared statements*) [Alsh06; Shif06]. W instrukcjach przygotowawczych znaki ucieczki wstawiane są automatycznie [Trac05, s.67], w wyniku czego, bez dodatkowego wysiłku, programiści mogą osiągnąć odpowiednio wysoki poziom bezpieczeństwa. Ważne jest, aby przy tworzeniu połączenia z bazą danych zdefiniować odpowiedni zestaw kodowania znaków.

Istotną cechą implementacji w PHP MySQL-owej funkcji wykonywania zapytań `mysql_query()`, jest to, że nie umożliwia ona przetworzenia więcej niż jednego zapytania na raz. To, co z punktu widzenia wydajności systemu, można uznać za wadę, w rozważaniach o bezpieczeństwie staje się niewątpliwą zaletą, ponieważ w znacznym stopniu ogranicza możliwości wykonania iniekcji wrogiego kodu SQL, uniemożliwiając napastnikowi dołączanie własnych zapytań.

Jak pokazano w tym punkcie ochrona przed *SQL Injection* wcale nie należy do zadań skomplikowanych, wystarczy pamiętać o kilku przedstawionych powyżej zasadach.

## Uprawnienia

System zarządzania bazą danych MySQL opisuje uprawnienia użytkowników do wykonywania poleceń SQL w formie list kontroli dostępu (ang. *Access Control List*) [Atki03, s.463]. Każdy

---

<sup>2</sup> Polski termin „zapytania preinterpretowane” został zaproponowany w pozycji książkowej [GBR05, s.170]. W literaturze można spotkać także określenia „instrukcje przygotowawcze” [Trac05, s.67] oraz „wyrażenia spreparowane” [Shem05, s.44].

użytkownik identyfikowany jest przez login (nazwę), hasło oraz serwer, z którego się łączy. Gdy następuje żądanie dostępu, system sprawdza czy na listach występują odpowiednie wpisy. Do przydzielania i odbierania uprawnień MySQL udostępnia standardowe instrukcje SQL, czyli GRANT (przydzielanie) i REVOKE (odbieranie).

Uprawnienia dostępu do danych można przydzielać na czterech różnych poziomach [West03] (wszystkie wspomniane niżej tabele przechowywane są w bazie danych mysql) :

- Globalnie – te uprawnienia obowiązują dla wszystkich baz na serwerze. Przechowywane w tabeli user.
- Baza danych – te uprawnienia obowiązują dla wszystkich tabel w bazie danych. Przechowywane w tabelach db (uprawnienia dla całej bazy danych) i host (uprawnienia dla wszystkich użytkowników z danego serwera).
- Tabela - te uprawnienia obowiązują dla wszystkich kolumn w danej tabeli. Przechowywane w tabeli tables\_priv (w bazie danych mysql).
- Kolumna - te uprawnienia obowiązują dla poszczególnych kolumn w tabeli. Przechowywane w tabeli columns\_priv (w bazie danych mysql).

Na każdym z tych poziomów mogą być przypisane różne uprawnienia (zob. tabela 1).

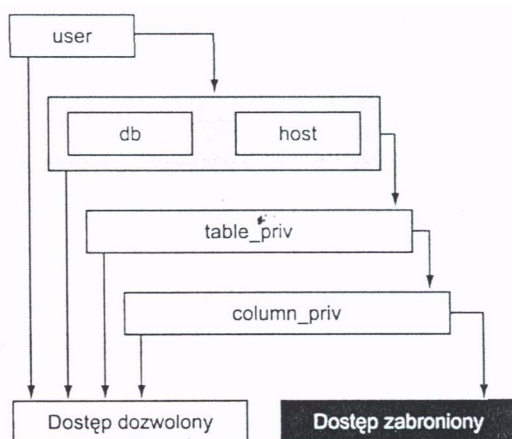
MySQL sprawdza uprawnienia dla każdej instrukcji przekazywanej przez użytkownika w sposób następujący [Atki03, s.468-469] :

- (1) Tabela user – sprawdzenie uprawnień globalnych. Zwykli użytkownicy nie mogą posiadać uprawnień globalnych, więc w tym przypadku następuje automatyczne przejście do kroku następnego.
- (2) Tabela db i host – w pierwszej kolejności sprawdzana jest tabela db. Gdy kolumna Host tej tabeli jest pusta, wtedy sprawdzana jest tabela host. Ostatecznym zestawem uprawnień jest wtedy iloczyn zbiorów uprawnień z obu tabel. Jeśli obie tabele nie opisują uprawnień dla danego użytkownika, następuje przejście do kroku kolejnego.
- (3) Tabela tables\_priv – sprawdzenie uprawnień dla poszczególnych tabel.
- (4) Tabela columns\_priv – sprawdzenie uprawnień dla poszczególnych kolumn tabeli.
- (5) Brak odpowiedniego wpisu dla danego użytkownika w żadnej z powyższych tabel powoduje odmowę wykonania żądanej instrukcji.

Uprawnienie	Nazwa kolumny	Kontekst	Opis
CREATE	Create_priv	baza danych, tabele, indeksy	Pozwala tworzyć tabele i bazy danych
DROP	Drop_priv	baza danych, tabele	Pozwala usuwać tabele i bazy danych
GRANT OPTION	Grant_priv	baza danych, tabele, procedury	Pozwala tworzyć i przekazywać uprawnienia innym.
REFERENCES	References_priv	baza danych, tabele	Brak implementacji. Istnieje dla zgodności składniowej ze standardem SQL
ALTER	Alter_priv	tabele	Pozwala używać polecenia ALTER TABLE
DELETE	Delete_priv	tabele	Pozwala usuwać rekordy z tabeli
INDEX	Index_priv	tabele	Pozwala dodawać i usuwać indeksy (CREATE INDEX / DROP INDEX)
INSERT	Insert_priv	tabele	Pozwala dodawać rekordy do tabeli
SELECT	Select_priv	tabele	Pozwala pobierać rekordy z tabeli
UPDATE	Update_priv	tabele	Pozwala aktualizować rekordy w tabeli
FILE	File_priv	dostęp do plików na serwerze	Pozwala korzystać z lokalnego systemu plików
CREATE TEMPORARY TABLES	Create_tmp_table_priv	administracja serwera	Pozwala tworzyć tabele tymczasowe.
LOCK TABLES	Lock_tables_priv	administracja serwera	Pozwala blokować te tabele, dla których użytkownik posiada uprawnienie SELECT.
PROCESS	Process_priv	administracja serwera	Pozwala wyświetlać listę procesów i procesy te kończyć
RELOAD	Reload_priv	administracja serwera	Pozwala czyścić bufor (FLUSH)
REPLICATION CLIENT	Repl_client_priv	administracja serwera	Pozwala pytać o położenie serwerów używanych w replikacji.
REPLICATION SLAVE	Repl_slave_priv	administracja serwera	Pozwala czytać binarne dzienniki zdarzeń z serwera nadrzędnego.
SHOW DATABASES	Show_db_priv	administracja serwera	Pozwala zobaczyć listę wszystkich baz danych.
SHUTDOWN	Shutdown_priv	administracja serwera	Pozwala wyłączać serwer.
SUPER	Super_priv	administracja serwera	Pozwala używać instrukcji CHANGE MASTER, KILL, PURGE MASTER LOGS, i SET GLOBAL, poleceń debugowania (mysqldadmin debug) oraz łączyć się z bazą nawet jeśli został wyczerpany limit połączeń (max_connections).

Tabela 1: Uprawnienia dostępne w MySQL 3.23, 4.0 i 4.1

Źródło: Opracowanie własne na podstawie [Atki03, s.467; WWW2].



Rysunek 1: Stosowanie uprawnień w MySQL

Źródło: [Atki03, s.468].

Podstawowe korzyści, jakie płyną z opisywanego w tym punkcie systemu kontroli praw dostępu, to [Luba03] :

- możliwość szczegółowego podziału zadań i odpowiedzialności,
- włamanie na jedno z kont nie oznacza całkowitej kontroli nad systemem.

Niestety, jeśli uprawnienia bazy danych mają być używane jako środek zabezpieczenia serwisu WWW, to powinno się dla każdego użytkownika serwisu założyć oddzielne konto w bazie danych, co jest często kłopotliwe w implementacji, a przede wszystkim prawie zawsze niewydajne [Cieb03, s.4]. Alternatywnym rozwiązaniem jest zarządzanie uprawnieniami w oparciu o role [West03]. Znika wtedy problem zbyt dużej liczby kont w systemie. W podstawowym modelu powinny być utworzone dwie role : administrator i użytkownik. W zależności od specyfiki systemu można tworzyć kolejne, ale już ten najprostrzy model daje zwiększony poziom bezpieczeństwa, który dla wielu witryn może być wystarczający.

Projektując system zarządzania uprawnieniami na poziomie bazy danych, pamiętać należy, że jeśli chodzi o bezpieczeństwo całego serwisu WWW, to z punktu widzenia aplikacji klienckiej, o wiele ważniejszą kwestią jest odpowiednia ochrona przed wstrzykiwaniem złośliwego kodu SQL. Uprawnienia bazodanowe nie będą bowiem potrzebne, jeśli aplikacja będzie dopuszczać do bazy danych tylko wiarygodne i sprawdzone zapytania. Dotyczy to głównie sytuacji, w których bezpośredni dostęp do bazy jest niemożliwy lub bardzo utrudniony, np. gdy aplikacja i baza danych znajdują się na tym samym serwerze lub w tej samej dobrze chronionej sieci lokalnej.

## Zapewnienie integralności danych

Integralność danych to gwarancja, że dane są zabezpieczone przed przypadkową lub umyślną (nieautoryzowaną) modyfikacją [MMVEM04, s.4]. Jest to własność przejawiająca się tym, że baza danych pozostaje poprawna w całym procesie wykonywania różnych operacji [Sund91, s.42]. Integralność można zapewnić poprzez kontrolę współbieżności oraz więzy spójności.

Kontrola wielodostępu (współbieżności) to proces zarządzania operacjami wykonywanymi jednocześnie na bazie danych bez dopuszczania do niepożądanych oddziaływań między nimi [CoBe04, s.36]. MySQL automatycznie obsługuje współbieżność dla pojedynczych zapytań, jednak w sytuacjach, gdy dwie lub więcej instrukcji muszą być wykonane za jednym razem, SQL używa transakcji lub blokad [Atki03, s.102].

Transakcja to jedna lub ciąg instrukcji SQL posiadających następujące cechy [Fowl05, s.72; SBP01, s.236 ]:

- Niepodzielność (ang. *atomicity*) – każda operacja należąca do transakcji musi zakończyć się powodzeniem. Jeśli wynikiem chociaż jednej operacji będzie błąd, odwołane muszą być wszystkie operacje należące do transakcji. Transakcja kończy się instrukcją COMMIT (zatwierdzenie) lub ROLLBACK (odrzućcie, wycofanie).
- Spójność (ang. *consistency*) – dane muszą być spójne i poprawne, zarówno w chwili rozpoczęcia transakcji, jak i w chwili jej zakończenia.
- Izolacja (ang. *isolation*) – zmiany dokonane w ramach transakcji stają się widoczne dla innych transakcji dopiero po jej zatwierdzeniu (wykonaniu COMMIT). W standardzie SQL wyjątkiem od tej zasady są poziomy izolacji READ UNCOMMITTED i READ COMMITTED.
- Trwałość (ang. *durability*) – wynik zatwierdzonej transakcji jest trwały, tj. zmiany są zapisywane na trwałe i stają się widoczne dla innych transakcji.

Teoretycznie system zarządzania relacyjną bazą danych powinien zapewnić pełną izolację pomiędzy transakcjami, jednak w praktyce odbywa się na kilku poziomach [Atki03, s.103]. Wyróżnia się cztery poziomy izolacji [SBP01, s.237-240; Fowl05 s.74]:

- Odczyt niezatwierdzonych (READ UNCOMMITTED) – tzw. „brudne czytanie” (ang. *dirty read*). Inna transakcja może czytać dane zmienione, mimo że nie zostały jeszcze zatwierdzone.
- Odczyt zatwierdzonych (READ COMMITTED) – możliwe jest aktualizowanie przez jedną transakcję danych wczytanych przez niezakończoną jeszcze drugą transakcję.
- Odczyt powtarzalny (REPEATABLE READ) – zabezpiecza przed modyfikacją wczytanych

danych, ale nie przed dołączaniem nowych wierszy (fantomów).

- Szeregowalność (SERIALIZABLE) – zapewnia najwyższy poziom izolacji. Chroni przed pojawieniem się fantomów.

Poziom izolacji	Brak odtwarzalności, „brudne czytanie”	Anomalia powtórnego czytania	Fantomy
Odczyt niezatwierdzonych (READ UNCOMMITTED)	TAK	TAK	TAK
Odczyt zatwierdzonych (READ COMMITTED)	NIE	TAK	TAK
Odczyt powtarzalny (REPEATABLE READ)	NIE	NIE	TAK
Szeregowalność (SERIALIZABLE)	NIE	NIE	NIE

Tabela 2: Związek poziomów izolacji z problemami przetwarzania transakcji  
Źródło: Opracowanie własne napodstawie [SBP01, s.237; Fowl05, s.74].

Przyjęcie konkretnego poziomu wiąże się z określonymi problemami (zob. tabela 2) – zbyt niski poziom może doprowadzić do niekorzystnych cech związanych z zachowaniem spójności danych, natomiast zbyt wysoki może powodować opóźnianie transakcji [SBP01, s.230]. Wybór zależy od wymaganego w danej chwili poziomu bezpieczeństwa. Domyślnym poziomem w bazie MySQL jest odczyt powtarzalny.

MySQL pracuje domyślnie w trybie autozatwierdzania (autocommit = 1), co oznacza, że wyniki każdej pojedynczej operacji mają charakter trwały, czyli stają się widoczne od razu po zatwierdzeniu. Dlatego konieczna jest zmiana wartości tego parametru albo skorzystanie z instrukcji START TRANSACTION lub BEGIN dla rozpoczęcia transakcji.

W najpopularniejszym silniku dla MySQL, MyISAM, zrezygnowano z implementowania mechanizmu obsługi transakcji. Jego twórcy doszli do wniosku, że zapewnienie poprawności i integralności danych leży w gestii programistów aplikacji [GrBu01, s.18]. Można polemizować z zasadnością tego stwierdzenia, ale nie zmienia to faktu, że brakującą funkcjonalność można uzyskać dzięki innemu silnikowi, InnoDB.

Innym sposobem na zapewnienie integralności danych są blokady. Blokady można zdefiniować jako procedury wykorzystywane do kontroli jednoczesnego dostępu do danych - gdy jedna transakcja odwołuje się do bazy danych, blokada może zapobiec dostępowi innych transakcji, które mogą

spowodować błędy [CoBe04, s.44]. Blokowanie jest wykorzystywane w transakcjach automatycznie, ale oprócz tego, MySQL pozwala na ustanawianie blokad przez użytkownika.

Jedną z możliwości jest użycie instrukcji LOCK TABLES. Pozwala ona na uzyskanie dwóch rodzajów blokad (ten typ blokad automatycznie stosuje także silnik InnoDB w zarządzaniu transakcjami) :

- Blokadę dzielonej (ang. *shared lock*) – jeżeli transakcji zostanie przyznana blokada dzielona jednostki danych, to transakcja może odczytywać wartość tej jednostki, ale nie może jej zmieniać [CoBe04, s.45]. Inne transakcje wtedy takżenie posiadają prawa do zapisu.
- Blokadę wyłącznej (ang. *exclusive lock*) – jeżeli transakcji zostanie przydzielona blokada wyłączna jednostki danych, to transakcja może zarówno odczytywać wartość tej jednostki, jak i ją zmieniać [CoBe04, s.45].

Wspomniana instrukcja umożliwia uzyskanie blokady tylko dla tabel, co uznać należy za jej główną wadę ze względu na prawdopodobne problemy wydajnościowe, wynikłe z czasowej (nawet krótkotrwałej) odmowy dostępu do całej tabeli dla innych użytkowników.

Blokady można także utworzyć stosując funkcje MySQL GET\_LOCK() i RELEASE\_LOCK(). Korzystanie z tych funkcji nazywa się blokowaniem z poziomu aplikacji, ponieważ nie są one połączone z żadnym zasobem, a znaczenie nadawane jest im w obrębie aplikacji – dzięki temu można symulować blokowanie o dowolnej dokładności, np. dla wierszy [Atki03, s.107].

Integralność danych gromadzonych w bazie danych można zapewnić także poprzez więzy spójności. Rozróżnia się dwa typy więzów : więzy spójności encji oraz więzy spójności referencyjnej.

Więzy spójności encji ograniczają możliwe wartości, jakie mogą się pojawić w wierszu tabeli [BaSt01, s.58]. W MySQL można zdefiniować następujące więzy [BaSt01, s.58] :

- Więzy klucza głównego PRIMARY KEY – wartości w określonych kolumnach jednoznacznie identyfikują wiersz.
- Więzy klucza jednoznacznego UNIQUE – wartości w określonych kolumnach jednoznacznie identyfikują wiersz. Różni się od PRIMARY KEY tym, że jest dopuszczalna wartość NULL oraz że jedna tabela może posiadać wiele kluczy tego typu.
- Więzy NOT NULL – wartości w określonych kolumnach nie mogą mieć wartości NULL.

Więzy spójności referencyjnej zapewniają, że zbiór wartości w kolumnach klucza obcego jest zawsze podzbiorem zbioru wartości odpowiadającego mu klucza głównego lub jednoznacznego [BaSt01, s.58]. Podobnie jak w przypadku transakcji, funkcjonalność ta zaimplementowana jest tylko w silniku InnoDB.

Zapewnienie integralności danych ma niewątpliwie duże znaczenie dla bezpieczeństwa całego systemu informatycznego, dlatego wybór innego typu tabel niż InnoDB, nie powinien być w ogóle brany pod uwagę. Ewentualnie wszelkie opisane tu zabezpieczenia mogą być szczegółowo wdrożone na poziomie aplikacji, jednak mając do dyspozycji ich natywną obsługę w InnoDB, nie jest to raczej dobrym i efektywnym rozwiązaniem.

## Literatura

- [Alsh06] Alshanevsky I.: *mysql\_real\_escape\_string() versus Prepared Statements*, Ilia Alshanevsky iBlog, 22.01.2006. [http://ilia.ws/archives/103-mysql\\_real\\_escape\\_string-versus-Prepared-Statements.html](http://ilia.ws/archives/103-mysql_real_escape_string-versus-Prepared-Statements.html)
- [Atki03] Atkinson L.: *MySQL*, Helion 2003.
- [BaSt01] Banachowski L., Stencel K.: *Bazy danych: Projektowanie aplikacji na serwerze*, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2001.
- [Cieb03] Ciebiera K.: *Filtrowanie zapytań SQL w środowisku aplikacji internetowych*, V Krajowa Konferencja Inżynierii Oprogramowania KKIO 2003, 14-17.10.2003, Szklarska Poręba, w: Huzar Z., Mazur Z.: *Problemy i metody inżynierii*
- [CoBe04] Connolly T.M., Begg C.E.: *Systemy baz danych : praktyczne metody projektowania implementacji i zarządzania, T.2*, RM, Warszawa 2004.
- [Fowl05] Fowler M.: *Architektura systemów zarządzania przedsiębiorstwem. Wzorce projektowe*, Helion 2005.
- [GBR05] Gutmans A., Bakken S.S., Rethans D.: *PHP5. Tajniki programowania*, Helion 2005.
- [GrBu01] Greenspan J., Bulger B.: *MySQL/PHP Database Applications*, M&T Books, 2001.
- [Luba03] Lubaczewski H.: *Bezpieczeństwo w PostgreSQL*, Wykład z serii GNU/Politechnika, 22.03.2003, Gliwice.
- [MMVEM04] Meier J.D., Mackman A., Vasireddy S., Escamilla R., Murukan A.: *Udoskonalanie zabezpieczeń aplikacji i serwerów internetowych*, Promise 2004.
- [OfSh04] Ofer M., Shulman A.: *SQL Injection Signatures Evasion*, Imperva 2004. [http://www.imperva.com/application\\_defense\\_center/white\\_papers/sql\\_injection\\_signatures\\_evasion.html](http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html)
- [SBP01] Stokłosa J., Bilski T., Pankowski T.: *Bezpieczeństwo danych w systemach informatycznych*, Wydawnictwo Naukowe PWN Warszawa-Poznań 2001.
- [Shem05] Shema M.: *Zaawansowane ataki SQL Injection*, Hakin9 nr 5/2005, s.38-45.
- [Shif06] Shiflett C.: *The addslashes() Versus mysql\_real\_escape\_string() Debate*, Chris Shiflett Blog, 22.01.2006. <http://shiflett.org/archive/184>
- [Sund91] Sundgren B.: *Bazy i modele danych*, PWE Warszawa 1991.
- [Trac05] Trachtenberg A.: *PHP5. Nowe możliwości*, Helion 2005.
- [West03] Westphal K.: *Secure MySQL Database Design*, SecurityFocus, 18-02-2003. <http://www.securityfocus.com/infocus/1667>
- [WWW1] *Podręcznik PHP*. <http://www.php.net/manual/pl/>
- [WWW2] *Dokumentacja MySQL 3.23, 4.0, 4.1*. <http://dev.mysql.com/doc/refman/4.1/en/>