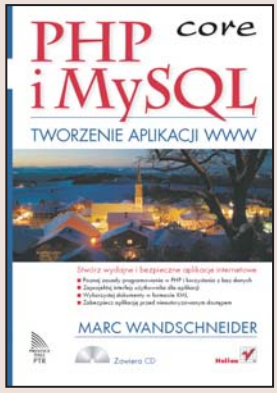
PHP i MySQL. Tworzenie aplikacji WWW



Autor: Marc Wandschneider
Tłumaczenie: Jarosław Dobrzański, Radosław Meryk
ISBN: 83-246-0323-9
Tytuł oryginału: Core Web Application Development with PHP and MySQL
Format: B5, stron: 856

Stwórz wydajne i bezpieczne aplikacje internetowe

- Poznaj zasady programowania w PHP i korzystania z baz danych
- Zaprojektuj interfejs użytkownika dla aplikacji
- Wykorzystaj dokumenty w formacie XML
- Zabezpiecz aplikację przed nieautoryzowanym dostępem

Sieć WWW już dawno przestała być jedynie zbiorem setek statycznych witryn. Dziś w sieci można znaleźć dziesiątki aplikacji – sklepów, katalogów, systemów bankowych, blogów i wielu innych. Do tworzenia takich aplikacji wykorzystuje się kilka technologii, z których największą popularnością cieszy się język PHP w połączeniu z bazą danych MySQL. Nieodpłatny dostęp, łatwość obsługi oraz potężne możliwości sprawiły, że ta platforma jest stosowana przez ogromne rzesze programistów aplikacji WWW na całym świecie.

„PHP i MySQL. Tworzenie aplikacji WWW" to książka, dzięki której poznasz możliwości tej technologii i nauczysz się z nich korzystać, pisząc aplikacje internetowe. Dowiesz się, jak tworzyć programy w języku PHP, manipulować danymi zgromadzonymi w bazie, projektować rozbudowane aplikacje i wdrażać je. Przeczytasz tu o zabezpieczaniu aplikacji, usuwaniu błędów, korzystaniu z plików XML i usług sieciowych oraz projektowaniu ergonomicznych interfejsów użytkownika. Dzięki praktycznym przykładom szybko nauczysz się stosować PHP i MySQL w swoich projektach.

- Programowanie w języku PHP
- Organizacja kodu
- Zasady programowania obiektowego
- Projektowanie i tworzenie baz danych
- Stosowanie języka SQL
- Przygotowywanie projektu aplikacji
- Budowanie interfejsu użytkownika
- Usuwanie błędów z kodu
- Metody uwierzytelniania użytkowników
- Wyrażenia regularne
- Usługi sieciowe i protokół SOAP
- Wdrażanie aplikacji

Rozpocznij przygodę z programowaniem w PHP

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

Spis treści

11

Wykorzystanie baz danych: zaawansowany dostęp do danych

W rozdziale 10. „Wykorzystanie baz danych: przechowywanie i pobieranie informacji" omówiłem język SQL w zakresie wykonywania podstawowych operacji na tabelach: wprowadzania danych, pobierania rekordów, ich aktualizowania i usuwania. W tym rozdziale wykorzystamy poznane podstawy i przejdziemy do bardziej zaawansowanych zagadnień i konstrukcji języka SQL.

W tym rozdziale:

- nauczymy się, gdzie, kiedy i jak korzystać z transakcji;
- dowiemy się, w jaki sposób wykonuje się zaawansowane zadania języka SQL, takie jak korzystanie z wyrażeń i funkcji;
- poznamy sposoby modyfikowania schematów tabeli w celu dodawania nowych kolumn lub zmiany nazw tabel.

Transakcje

Podczas wykonywania operacji na danych zawsze trzeba dbać o zachowanie spójności i integralności danych. O ile system zarządzania bazą danych zapewnia bezpieczne wprowadzanie danych do tabel i pobieranie z nich informacji, nie może zagwarantować, że dane zapisane w bazie zawsze będą miały sens. Przykładowo, jeśli zaznaczyliśmy w tabeli zawierającej informacje o książkach, że sprzedaliśmy trzy książki klientowi, ale przed zapisaniem zamówienia w odpowiedniej tabeli nastąpiła awaria zasilania, baza danych będzie zawierała niespójne informacje. Może również wystąpić problem rywalizacji, kiedy dwóch użytkowników jednocześnie spróbuje kupić ostatni egzemplarz książki. W najgorszym przypadku może się zdarzyć, że zamówienie złożą obaj użytkownicy.

Wspomniane problemy można rozwiązać za pomocą odpowiedniego kodu, opracowując mechanizmy „blokowania" tabel w celu uniemożliwienia korzystania z nich innym użytkownikom. Można również utworzyć mechanizmy wykrywające niespójności i niedokończone operacje. Jest to jednak sposób kosztowny, skomplikowany i podatny na błędy. Lepsze efekty można osiągnąć powierzając te działania serwerowi bazy danych za pomocą **transakcji**.

Problem

Przeanalizujmy dokładniej przykład księgarni online. Wyobraźmy sobie, że utworzono prymitywną tabelę zawierającą dane o wszystkich produktach przeznaczonych na sprzedaż oraz tabelę opisującą zamówienia. Dla uproszczenia zakładamy, że można zakupić tylko jeden typ książek. W przykładzie wykorzystamy zdefiniowane poniżej tabele i pominiemy wiele szczegółów zamówień, takich jak informacje o wysyłce, cenie oraz płatnościach.

```
CREATE TABLE Products
(
    pid INTEGER AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    isbn VARCHAR(200) NOT NULL,
    price NUMERIC(10,2) NOT NULL,
    number_in_stock INTEGER NOT NULL
)
ENGINE = InnoDB;

CREATE TABLE Orders
(
    order_id INTEGER AUTO_INCREMENT PRIMARY KEY,
    order_date DATETIME NOT NULL,
    user_id INTEGER NOT NULL,
    product INTEGER NOT NULL,
    num_units INTEGER NOT NULL,
    FOREIGN KEY(user_id) REFERENCES Users(user_id),
    FOREIGN KEY(product) REFERENCES Products(pid)
)
ENGINE = InnoDB;
```

W przypadku sprzedaży książki klientowi powinniśmy wykonać dwa zapytania:

```
UPDATE Products SET number_in_stock = 10 WHERE pid = 343;
INSERT INTO Orders (order_date, user_id, product, num_units)
    VALUES (NOW(), 4358, 343, 1);
```

Problem powstanie w przypadku, gdy pierwsze zapytanie wykona się pomyślnie, ale drugie z jakichkolwiek przyczyn nie zostanie wykonane. Taka sytuacja może się zdarzyć z któregokolwiek z następujących powodów:

- utracono połączenie z bazą danych;
- nastąpiła awaria serwera bazy danych bądź brakło miejsca na dysku twardym, co przyczyniło się do niepowodzenia drugiego zapytania;
- z powodu awarii zasilania nastąpiło wyłączenie serwera.

Wielu Czytelników zapewne wzrusza w tej chwili ramionami, mówiąc „taka sytuacja nie może mi się **przytrafić**". Jeśli jednak weźmiemy za przykład duże aplikacje internetowe, przetwarzające tysiące, jeśli nie miliony transakcji dziennie, przekonamy się, że takie sytuacje zdarzają się, a klienci oczekują, że powstałe problemy zostaną rozwiązane w rozsądny i niekłopotliwy sposób. W powyższym przykładzie na szczęście klient nie został obciążony za coś, czego nie otrzyma, ale byłoby lepiej, gdyby aplikacja mogła automatycznie wykrywać awarie i odtwarzać brakujące pozycje w polu `number_in_stock` tabeli `Products`.

Rozwiązanie

Mając na uwadze możliwe problemy, wprowadzimy pojęcie transakcji. Jest to mechanizm pozwalający na grupowanie wielu zapytań i instrukcji SQL w jedną niepodzielną operację w bazie danych. Albo jest wykonywana i akceptowana (**zatwierdzana**) cała grupa instrukcji, albo żadna z nich i wtedy skutki operacji są **wycofywane**. W rzeczywistości taki mechanizm jest bardziej skomplikowany niż się wydaje na pierwszy rzut oka. System zarządzania relacyjną bazą danych obsługujący transakcje musi spełniać tzw. reguły `ACID`:

- **Niepodzielność** (ang. *Atomicity*) — transakcje rządzą się zasadą **wszystko albo nic**. Cała grupa działań w obrębie transakcji musi być traktowana jako niepodzielna jednostka. W przypadku niepowodzenia żadna z operacji nie może się wykonać.
- **Spójność** (ang. *Consistency*) — po wykonaniu transakcji baza danych musi znajdować się w spójnym stanie, a wszystkie ograniczenia i reguły integralności danych muszą zostać spełnione.
- **Izolacja** (ang. *Isolation*) — zmiany dokonywane przez transakcję w trakcie jej przeprowadzania nie mogą być dostępne dla innych transakcji. Wewnętrzne operacje różnych transakcji powinny być od siebie odseparowane.
- **Trwałość** (ang. *Durability*) — serwer musi mieć możliwość zakończenia transakcji w trybie nadzwyczajnym. Może to oznaczać, że po pomyślnie zakończonej transakcji pozostaną zaległe operacje lub — jeśli transakcja zostanie przerwana lub anulowana — tylko część operacji zostanie wykonana. W każdym przypadku serwer musi mieć możliwość natychmiastowego zakończenia działania i rekonstrukcji spójnego stanu przy ponownym uruchomieniu (niezależnie od tego, czy mamy do czynienia z zatwierdzoną transakcją, czy z transakcją anulowaną).

Wszystkie bazy danych omawiane w tej książce obsługują transakcje. W systemie MySQL trzeba jednak zwrócić uwagę na wykorzystywany typ tabeli. Jeśli tabele zostały utworzone z wykorzystaniem mechanizmu zapisu MyISAM, transakcje nie będą obsługiwane. W tej książce wszędzie tam, gdzie będziemy wykorzystywać transakcje, zastosujemy mechanizm zapisu InnoDB.

Niektóre serwery obsługują różne **poziomy izolacji transakcji**. Większość obsługuje cztery poziomy. Na najniższym poziomie (który jednocześnie jest najszybszy) transakcje mają możliwość odczytywania postępu i bieżącego stanu innych transakcji. Na najwyższym poziomie transakcje są całkowicie od siebie odseparowane (poziom najwolniejszy). Istnienie tych poziomów można potraktować jako przyznanie się do potrzeby kompromisu pomiędzy poziomem izolacji transakcji a wydajnością aplikacji. Dla naszych potrzeb odpowiedni poziom izolacji zapewnia poziom domyślny dla większości serwerów baz danych omawianych

w tej książce (**powtarzalny odczyt** — ang. *repeatable read*). Wspomnę jeszcze, że w systemie Oracle domyślnie obowiązuje nieco niższy poziom izolacji, ale to nie stanowi dla nas problemu. Czytelników zainteresowanych bardziej szczegółowym zapoznaniem się z obsługą transakcji polecam sięgnięcie do książki, która w bardziej obszerny sposób opisuje ten temat.

Pisanie transakcji

Sposób oznaczania początku transakcji jest różny dla różnych serwerów baz danych, jednak w przypadku większości z nich transakcje rozpoczyna się prostą instrukcją:

```
BEGIN;
```

W niektórych systemach baz danych jest dostępna opcja **automatycznego zatwierdzania** (ang. *autocommit*) zarządzająca sposobem wykonywania zapytań. Jeśli ustawi się ją na wartość `TRUE` (lub 1, co jest wartością domyślną), wszystkie wprowadzane zapytania będą automatycznie zatwierdzane w bazie danych. W przypadku ustawienia jej na `FALSE` (0), zapytania będą rejestrowane tak, jakby wchodziły w skład transakcji. W tym przypadku zatwierdzanie wykonuje się za pomocą jawnej instrukcji. Tak więc w systemie MySQL można rozpocząć transakcję za pomocą instrukcji:

```
SET AUTOCOMMIT =0;
```

Po zakończeniu wykonywania wszystkich zapytań w transakcji, jej wyniki zatwierdza się w bazie danych za pomocą następującej instrukcji:

```
COMMIT;
```

Aby anulować transakcję i cofnąć wykonane w niej operacje, należy zastosować następującą instrukcję:

```
ROLLBACK;
```

W przypadku zakończenia połączenia z serwerem, jego awarii lub przerwania transakcji w inny sposób przed wykonaniem instrukcji `COMMIT`, automatycznie wykonuje się instrukcja `ROLLBACK` i transakcja jest anulowana.

Wróćmy do naszego problemu sprzedaży książek. Jeśli skorzystamy z transakcji, sekwencja instrukcji SQL przyjmie następującą postać:

```
BEGIN;

UPDATE Products SET  number_in_stock = 10 WHERE pid = 343;

INSERT INTO Orders (order_date, user_id, product, num_units)
    VALUES(NOW(), 4538, 343, 1);

Jeśli doszliśmy do tego miejsca bez błędów:
COMMIT;

W innym przypadku, jeśli wystąpiły błędy:
ROLLBACK;
```

Bardziej złożony problem

Trudno się do tego przyznać, ale w naszym systemie ciągle jeszcze występuje pewien kłopot. Rozwiązaliśmy problem spójności bazy danych w przypadku awarii serwera podczas wykonywania programu sprzedaży, ale może się zdarzyć inna sytuacja — kilku użytkowników będzie chciało zakupić jedną książkę w tym samym czasie.

Spróbujmy przeanalizować sytuację, w której dwóch użytkowników spróbuje zakupić tę samą książkę, podczas gdy w magazynie pozostał tylko jeden egzemplarz.

```
+-----+-------------+------+-------+-----------------+
| pid | title       | isbn | price | number_in_stock |
+-----+-------------+------+-------+-----------------+
| 343 | 'Szczęście' | 'xx' | 19.99 |               1 |
+-----+-------------+------+-------+-----------------+
```

Kod obsługi sprzedaży jest następujący:

```
BEGIN

SELECT number_in_stock FROM Products WHERE pid = 343;

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")

UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
    VALUES (NOW(), 4538, 343, 1);
Jeśli doszliśmy do tego miejsca bez błędów:
COMMIT;

W innym przypadku, jeśli wystąpiły błędy:
ROLLBACK;
```

Nasz nowy problem wystąpi w przypadku, gdy dwóch użytkowników naszej aplikacji próbuje zakupić tę samą książkę w tym samym czasie. Poniżej zaprezentuję sekwencję operacji wykonywanych przez obu użytkowników w przybliżeniu w tym samym czasie.

Użytkownik 1. rozpoczyna proces zakupu. Wykonują się następujące instrukcje:

```
[Użytkownik 1]
BEGIN

SELECT number_in_stock FROM Products WHERE pid = 343;

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")
```

Program wykonywany przez pierwszego użytkownika stwierdza, że w magazynie jest jeden egzemplarz książki i jest gotowy do obsługi zakupu. Jednak w tym samym czasie robi zakupy drugi użytkownik i wykonuje następujący kod:

```
[Użytkownik 2]
BEGIN

SELECT number_in_stock FROM Products WHERE pid = 343;

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")
```

Zatem program wykonywany przez drugiego użytkownika również stwierdza, że w magazynie pozostał jeden egzemplarz i jest gotowy do obsługi zakupu. Domyślny poziom izolacji transakcji dla naszego serwera bazy danych nie jest najbardziej restrykcyjny, zatem równolegle wykonujące się transakcje mogą odczytać wartości tego samego wiersza. Teraz jednak kod obsługujący pierwszego użytkownika wykona następujące instrukcje:

```
[Użytkownik 1.]

UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
    VALUES (NOW(), 4538, 343, 1);
COMMIT;
```

Użytkownik numer 1 pomyślnie zakupił książkę. Kiedy drugi użytkownik spróbuje ją zakupić za pomocą następującego kodu:

```
[Użytkownik 2.]

UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
    VALUES (NOW(), 4538, 343, 1);
COMMIT;
```

wykonanie zapytania UPDATE powiedzie się, ale **nie spowoduje aktualizacji wierszy w tabeli**! Wynika to z faktu, iż program obsługujący transakcję serwera bazy danych stwierdzi, że dane uległy zmianie i nie pozwoli drugiemu procesowi ich zmodyfikować. To jednak nie powoduje zgłoszenia błędu. W kodzie musimy teraz wprowadzić dodatkowe instrukcje, które wykryją, czy interesujący nas wiersz został zmodyfikowany i w takiej sytuacji anulują transakcję lub ponowią próbę aktualizacji, jeśli modyfikacja nie nastąpiła.

Istnieje jednak o wiele bardziej eleganckie rozwiązanie, polegające na zastosowaniu zmodyfikowanej instrukcji SELECT: SELECT... FOR UPDATE. W przypadku pobrania wartości wiersza za pomocą tego zapytania jednocześnie wskazujemy, że mamy zamiar zmodyfikować w nim dane. W tej sytuacji inne transakcje lub użytkownicy próbujący uzyskać dostęp do danych zostaną zablokowani do czasu zakończenia transakcji. Teraz możemy przepisać nasz kod w następujący sposób:

```
BEGIN

SELECT number_in_stock FROM Products
    WHERE pid = 343 FOR UPDATE;

Odjęcie 1 od wartości number_in_stock i ustawienie jej na nową wartość (tu oznaczymy ją jako "nowa")

UPDATE Products SET number_in_stock = nowa WHERE pid = 343;

INSERT INTO Orders(order_date, user_id, product, num_units)
    VALUES (NOW(), 4538, 343, 1);
Jeśli doszliśmy do tego miejsca bez błędów:
COMMIT;

W innym przypadku, jeśli wystąpiły błędy:
ROLLBACK;
```

Dzięki wprowadzeniu klauzuli FOR UPDATE wszystkie inne procesy próbujące uzyskać dostęp do tej samej wartości będą musiały czekać na wywołanie instrukcji COMMIT lub ROLLBACK. To eliminuje problem rywalizacji o zakup ostatniej książki.

Dla transakcji istnieją bardziej zaawansowane mechanizmy blokowania, ale w tej książce nie będziemy pisali na tyle skomplikowanych programów, aby mogły być potrzebne. Przykłady transakcji zaprezentuję w części V „Przykładowe projekty i dalsze pomysły".

Bardziej zaawansowane zapytania

Do tej pory omówiłem podstawowe zapytania SQL, jednak możliwości języka są o wiele większe. W tym podrozdziale opiszę niektóre bardziej zaawansowane własności języka SQL.

Łączenie wyrażeń

Słowo kluczowe WHERE umożliwia wprowadzenie warunków ograniczających zakres zwracanych wierszy przez niektóre zapytania, w tym SELECT, UPDATE i DELETE. Wcześniej używaliśmy prostych wyrażeń o postaci:

```
NazwaKolumny operator wartość
```

Wyrażenia można jednak łączyć ze sobą i w ten sposób formułować bardziej złożone warunki. Do tego celu służą słowa kluczowe AND i OR. Ich działanie jest podobne do działania operatorów AND i OR w innych językach programowania. Wynik operacji AND jest prawdziwy, jeśli są spełnione warunki po obu stronach operatora, natomiast wynik operacji OR jest prawdziwy, jeśli jest spełniony którykolwiek warunek po dowolnej stronie operatora. Dzięki zastosowaniu operatorów AND i OR można dowolnie formułować zapytania. Na przykład, aby znaleźć wszystkich użytkowników o imieniu Grażyna, którzy urodzili się po roku 1980, można wykorzystać następujące zapytanie:

```
SELECT user_name, user_email FROM Users
    WHERE full_name LIKE 'Grażyna%'
        AND birthdate >= '1980-01-01';
```

W podobny sposób można znaleźć użytkowników o imieniu Bogdan lub Bogusław:

```
SELECT user_name, user_email FROM Users
    WHERE full_name LIKE 'Bogdan%'
    OR full_name LIKE 'Bogusław%';
```

W połączeniach wielu wyrażeń tego typu można i należy stosować nawiasy, które objaśniają kolejność sprawdzania warunków:

```
SELECT user_name, full_name, birthdate FROM Users
    WHERE (full_name LIKE 'Bogdan%' OR full_name LIKE 'Bogusław%')
        AND birthdate >= '1980-01-01';
```

Określanie zbiorów i zakresów wartości

Do określania zakresów wartości w wyrażeniach w języku SQL są dostępne dwa przydatne słowa kluczowe: `IN` i `BETWEEN`.

Słowo kluczowe `IN` umożliwia określenie zbioru wartości skalarnych (nie można stosować symboli wieloznacznych), do którego ma należeć wartość w kolumnie:

```
SELECT * FROM Messages
    WHERE forum_id IN (1, 3, 7, 4);
```

Słowo kluczowe `BETWEEN` umożliwia zdefiniowanie zakresu wartości. Można je wykorzystać zarówno dla liczb, jak i dla dat:

```
SELECT * FROM Users
    WHERE birthdate BETWEEN '1970-01-01' AND '1970-12-31';
SELECT * FROM Messages
    WHERE message_id BETWEEN 1000 AND 5000;
```

W przypadku ciągów znaków jest nieco gorzej. Jak wspominałem wcześniej, do określenia zakresu ciągów znaków serwer bazy danych wykorzystuje zdefiniowany porządek sortowania. Dzięki zdefiniowaniu porządku sortowania można określić sposób porządkowania nie tylko danych w języku angielskim, ale także w innych językach. Tak więc następujące zapytanie:

```
SELECT * FROM Users
    WHERE user_name BETWEEN 'a' AND 'm';
```

zwraca nazwy użytkowników pomiędzy *a* a *m*. Przy wprowadzeniu tego zapytania może wyświetlić się następujący komunikat o błędzie:

```
ERROR 1270 (HY000): Illegal mix of collations
    (utf8_general_ci, IMPLICIT), (latin1_swedish_ci, COERCIBLE),
    (latin1_swedish_ci, COERCIBLE) for operation 'BETWEEN';
```

Najbardziej prawdopodobną przyczyną tego błędu jest wykorzystanie w programie klienckim używanym do połączenia z serwerem innego zestawu znaków i porządku sortowania niż ustawiony na serwerze (patrz rozdział 9. „Projektowanie i tworzenie baz danych"). Zazwyczaj na serwerze ustawia się kodowanie Unicode (utf8). Aby wyeliminować problem, należy ustawić w programie klienckim ten sam zestaw znaków i porządek sortowania, jak w bazie danych. W przypadku klienta mysql bazy danych MySQL robi się to za pomocą następującej instrukcji:

```
mysql> set character_set_connection = @@character_set_database;
```

Powyższa instrukcja ustawia także porządek sortowania na domyślny dla wybranego zestawu znaków. Porządek sortowania można również wprowadzić w osobnej instrukcji w następujący sposób:

```
mysql> set collation_connection = @@collation_database;
```

Dodatkową komplikacją jest fakt, iż niektóre serwery baz danych interpretują zakresy wartości zdefiniowane w klauzuli `BETWEEN` włącznie z wartościami granicznymi, natomiast inne nie uwzględniają tych wartości. Przed skorzystaniem ze słowa kluczowego `BETWEEN` należy upewnić się, w jaki sposób są interpretowane granice zakresu (we wszystkich bazach danych omawianych w tej książce granice przedziałów wchodzą w skład zakresu).

Pobieranie niepowtarzalnych wartości

Czasami zamiast wszystkich wartości w kolumnie (gdzie mogą występować duplikaty) interesuje nas tylko zbiór możliwości. W przypadku naszej tabeli `Messages` wykonanie zapytania:

```
SELECT forum_id FROM Messages;
```

może spowodować wyświetlenie wielu powtórzeń nazw forów, na których publikowano więcej niż jedną wiadomość. Jeśli jednak interesuje nas lista forów, na których opublikowano co najmniej jedną wiadomość, możemy w języku SQL zastosować zapytanie `SELECT DISTINCT`:

```
SELECT DISTINCT forum_id FROM Messages;
```

Powyższe zapytanie zwróci pojedyncze wartości pola `forum_id` i wyeliminuje wszystkie duplikaty.

Wykorzystanie funkcji SQL w zapytaniach

Oprócz wartości skalarnych i kolumn, w zapytaniach języka SQL można wykorzystać szereg funkcji. Mogą to być zarówno wbudowane funkcje serwera, jak i funkcje zdefiniowane przez użytkownika w systemach obsługujących tzw. **procedury składowane** (ang. *stored procedures*) — funkcje zdefiniowane przez użytkownika i przechowywane na serwerze w postaci skompilowanej.

W języku SQL występują dwie klasy funkcji:

- **Funkcje agregacji** — funkcje, które przetwarzają zbiór wartości (na przykład wartości danych w kolumnie tabeli) i zwracają pojedynczą wartość skalarną. Przykładem mogą być funkcje obliczające sumę wartości w kolumnie lub ich średnią arytmetyczną.
- **Funkcje skalarne** — funkcje przetwarzające pojedynczą wartość skalarną (na przykład wartość określonej kolumny w określonym wierszu) i zwracające pojedynczą wartość skalarną. W przypadku podania jako parametru nazwy kolumny, funkcja zwraca kolumnę, której wartości są jej wynikami dla wartości kolumny źródłowej. Przykładami mogą być funkcje zamiany ciągów znaków w polu na małe litery, konwersji walut oraz funkcje formatujące dane przed wyświetleniem.

W wyrażeniach można stosować wyłącznie funkcje skalarne. Funkcje agregacji można wykorzystać do przetwarzania wyników zapytania przed ich zwróceniem do użytkownika. Najczęściej stosuje się je w odniesieniu do wyników instrukcji `SELECT`.

Niestety, funkcje stanowią kolejny obszar, w którym poszczególne serwery baz danych bardzo się różnią pomiędzy sobą. Choć zestaw własności funkcjonalnych jest podobny, nazwy funkcji i sposób ich używania bardzo się różnią pomiędzy poszczególnymi serwerami. W tym rozdziale postaram się opisać najpopularniejsze funkcje dla najpopularniejszych serwerów. W przypadku, gdy rozbieżności będą zbyt wielkie, przedstawię wersję dla bazy danych MySQL (zwięzły opis odpowiedników dla innych serwerów baz danych można znaleźć w dodatku B „Odpowiedniki funkcji obsługi baz danych").

W przypadku wywoływania funkcji w zapytaniach SQL, pomiędzy nazwą funkcji a otwierającym znakiem nawiasu przed listą argumentów nie może być spacji.

```
AVG(daily_precip)       Dobrze
AVG  (daily_precip)     Źle
```

Funkcje numeryczne

Rozpocznę od opisania kilku funkcji numerycznych, ponieważ są one najbardziej zrozumiale i najłatwiejsze w użyciu. Wszystkie opisane poniżej funkcje to funkcje agregacji, które wykorzystamy do przetwarzania zbioru wierszy w zapytaniu w celu uzyskania pojedynczej wartości.

COUNT

Funkcję `COUNT` można wykorzystać na dwa sposoby: `COUNT(`*`NazwaKolumny`*`)` lub `COUNT(*)`. Pierwszy sposób wywołania zlicza w zestawie wyników liczbę wartości określonej kolumny różnych od NULL. W drugiej wersji funkcja `COUNT` zlicza liczbę wierszy w zestawie wyników. Na przykład instrukcja:

```
SELECT COUNT(*) FROM Users;
```

zlicza użytkowników w tabeli `Users`. Aby policzyć użytkowników w tabeli `Users`, których nazwiska są różne od `NULL` (co przy naszym schemacie tabeli jest dozwolone), można wykorzystać następujące zapytanie:

```
mysql> SELECT COUNT(full_name) from Users;
+------------------+
| count(full_name) |
+------------------+
|                4 |
+------------------+
1 row in set (1.26 sec)
```

SUM

Aby w zestawie wyników uzyskać sumę wartości zawartych w określonej kolumnie, można skorzystać z funkcji `SUM`. Gdybyśmy mieli tabelę z informacjami o pogodzie w określonym miejscu i chcieli obliczyć całkowitą wartość opadów w 2002 roku, moglibyśmy wykonać następujące zapytanie:

```
SELECT SUM(daily_precip) FROM DailyWeatherReports
    WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

MAX i MIN

Aby obliczyć maksymalną lub minimalną wartość kolumny w zestawie wyników, można skorzystać z funkcji `MAX` i `MIN`. Kontynuując nasz przykład bazy danych o pogodzie z poprzedniego punktu, aby znaleźć dni w 2002 roku, w których wystąpiły największe i najmniejsze opady, możemy skorzystać z następujących zapytań:

```
SELECT MAX(daily_precip) FROM DailyWeatherReports
    WHERE date BETWEEN '2002-01-01' AND '2002-12-31';

SELECT MIN(daily_precip) FROM DailyWeatherReports
    WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

Powyższe dwa zapytania można połączyć w jedno. Zestaw wyników takiego zapytania będzie składał się z dwóch kolumn:

```
SELECT MAX(daily_precip), MIN(daily_precip)
      FROM DailyWeatherReports
    WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

AVG

Aby obliczyć średnią arytmetyczną wartości w kolumnie, można skorzystać z funkcji `AVG`. Wykorzystuje się ją w następujący sposób:

```
SELECT AVG(daily_precip) FROM DailyWeatherReports
    WHERE date BETWEEN '2002-01-01' AND '2002-12-31';
```

Funkcje znakowe

W języku SQL występuje wiele przydatnych funkcji przetwarzania ciągów znaków. Większość z nich to funkcje skalarne, które można wykorzystać w wielu przypadkach.

Wydzielanie podciągów

Aby wydzielić część ciągu znaków w języku SQL, można skorzystać z funkcji `SUBSTRING` (w systemie Oracle — `SUBSTR`). Funkcja pobiera trzy argumenty: wartość (nazwę kolumny), z której ma być wydzielony podciąg, indeks pierwszego znaku podciągu oraz liczbę znaków do wydzielenia.

> **Uwaga**
>
> W odróżnieniu od języka PHP, gdzie indeksy mają wartości począwszy od 0, indeksy w ciągach znaków w języku SQL liczy się od 1. Oznacza to, że pierwszy znak w ciągu ma indeks 1.

Aby pobrać pierwszych 5 znaków z każdego wiersza tabeli zawierającej nazwy wszystkich stanów w USA i prowincji w Kanadzie, można wykonać następujące zapytanie:

```
SELECT SUBSTRING(name, 1, 5) FROM states_provinces;
```

Aby znaleźć stany i prowincje, których nazwy zaczynają się od sekwencji `New`, można wykonać następujące zapytanie:

```
SELECT * FROM states_provinces
    WHERE SUBSTRING(name, 1, 3) = 'New';
```

Konwersja wielkości liter

Aby przekształcić ciągi znaków na małe bądź wielkie litery, można skorzystać z funkcji LOWER lub UPPER.

```
SELECT LOWER(user_name) FROM Users;
SELECT UPPER(last_name) FROM customers;
```

Funkcji LOWER i UPPER rzadko używa się w wyrażeniach formułowanych w klauzuli WHERE, ponieważ — jak wspominałem w poprzednim rozdziale — w przypadku większości operacji sortowania i porównań wielkość liter nie ma znaczenia.

Wyszukiwanie ciągu znaków

Często przydaje się funkcja, która wyszukuje pozycję określonego ciągu znaków w tekście (lub kolumnie). Chociaż we wszystkich serwerach baz danych, które będziemy wykorzystywać, taka funkcja jest dostępna, nazwy i sposoby korzystania z niej mogą być różne. Zarówno w systemie MySQL, jak i Oracle jest dostępna funkcja INSTR:

```
INSTR(szukaj_tu, znajdź_mnie)
```

Na przykład, poniższe zapytanie można wykorzystać do pobrania imion użytkowników naszej internetowej tablicy ogłoszeń (przy założeniu, że w polu name imię występuje jako pierwsze):

```
SELECT user_name
       SUBSTRING(full_name, 1, INSTR(full_name, ' '))
  FROM Users;
```

Jeśli szukany ciąg nie zostanie odnaleziony, funkcja INSTR zwraca wartość 0.

Konkatenacja ciągów znaków

W języku SQL istnieje funkcja służąca do konkatenacji ciągów znaków. W różnych serwerach baz danych funkcja ta może nazywać się inaczej. W tym miejscu opiszę składnię polecenia w systemie MySQL:

```
CONCAT(wartość1, wartość2, ..., wartośćn)
```

Aby na podstawie zawartości tabeli Users uzyskać sformatowany ciąg znaków składający się z nazwy użytkownika i adresu e-mail, można wykonać następujące zapytanie:

```
SELECT CONCAT('Nazwa użytkownika: ',
              user_name,
              '\tAdres e-mail:',
              user_email)
    FROM Users;
```

Przycinanie ciągów znaków

Funkcja TRIM usuwa wiodące i końcowe spacja z ciągów znaków. Działa tak samo jak odpowiadająca jej funkcja w PHP:

```
SELECT user_name, TRIM(full_name), user_email
    FROM Users
    WHERE user_name LIKE 'F%';
```

Funkcje przetwarzania dat i godzin

W języku SQL występuje kilka przydatnych funkcji przetwarzających daty i godziny. Nie są one standardowe i w różnych serwerach występują pod innymi nazwami, ale większość podstawowych funkcji jest dostępna we wszystkich najpopularniejszych systemach baz danych.

Now

Aby odczytać bieżącą datę i godzinę w systemie MySQL i PostgreSQL, można skorzystać z funkcji NOW.

```
SELECT NOW();

INSERT INTO Orders (prodid, user_id, when)
    VALUES(445455423, 32345, Now());
```

Rok, miesiąc, dzień

W języku SQL występuje grupa funkcji służących do pobierania części wartości z dat. Funkcje te znacznie się różnią w różnych systemach baz danych. W bazie danych MySQL funkcja YEAR pobiera datę i na jej podstawie zwraca czterocyfrowy numer roku. Funkcja MONTH zwraca numer miesiąca daty, natomiast funkcje DAYOFMONTH, DAYOFWEEK i DAYNAME zwracają informacje na temat dnia w różnym formacie.

Formatowanie dat i godzin

Chociaż we wszystkich systemach baz danych występują funkcje służące do formatowania dat i godzin, poszczególne implementacje bardzo się różnią pomiędzy sobą. W systemie MySQL do formatowania dat służy funkcja DATE_FORMAT, która pobiera wartość daty do sformatowania oraz ciąg formatu.

```
DATE_FORMAT(sformatuj_mnie, ciąg_formatu)
```

Parametr *ciąg_formatu* to sekwencja znaków, które są zastępowane odpowiednimi wartościami dotyczącymi daty. Na przykład:

```
mysql> SELECT full_name, DATE_FORMAT(birthdate, '%W %D %M %Y')
    -> FROM Users
    -> WHERE birthdate <> '000-00-00';
+------------------------------+--------------------------------------+
| full_name                    | DATE_FORMAT(birthdate, '%W %D %M %Y') |
+------------------------------+--------------------------------------+
| Krzysztof Malinowski         | Tuesday 16th December 1980           |
| Akira Tanaka                 | Wednesday 13th September 0000        |
| Patrycja Dominikowska        | Monday 31st March 1975               |
+------------------------------+--------------------------------------+
3 rows in set (0.00 sec)
```

Najpopularniejsze i najbardziej interesujące kody formatów zestawiono w tabeli 11.1. Pełną listę wartości można znaleźć w dokumentacji bazy danych MySQL.

Tabela 11.1. *Kody formatów funkcji DATE_FORMAT*

Kod	Znaczenie
`%W`	Dzień tygodnia w języku serwera bazy danych (często jest nim język angielski).
`%w`	Dzień tygodnia w formacie liczbowym (0= niedziela, 6= sobota).
`%Y`	Rok według kalendarza juliańskiego przedstawiony za pomocą 4 cyfr (na przykład 1999).
`%y`	Rok według kalendarza juliańskiego przedstawiony za pomocą 2 cyfr (na przykład 33).
`%M`	Nazwa miesiąca w języku serwera bazy danych (często jest nim język angielski).
`%m`	Numer miesiąca w formacie liczbowym (01 – 12).
`%D`	Dzień miesiąca z przyrostkiem liczebnika porządkowego dla języka angielskiego (1st, 2nd, 3rd itd.).
`%d`	Dzień miesiąca w formacie liczbowym (01 – 31)
`%H`	Godzina w formacie 24-godzinnym (00 – 23).
`%h`	Godzina w formacie 12-godzinnym (01 – 12).
`%p`	Przyrostek AM bądź PM.
`%i`	Minuta w formacie liczbowym (00 – 59).
`%S` lub `%s`	Sekunda w formacie liczbowym (00 – 59).

Grupowanie danych dla funkcji agregacji

Funkcje agregacji w języku SQL są jeszcze bardziej przydatne, jeśli przed wykonaniem funkcji pogrupuje się wartości danych w tabeli. W zaprezentowanym poprzednio przykładzie bazy danych z informacjami o pogodzie pokazałem, w jaki sposób można wykorzystać funkcję `AVG` do obliczenia średniej wartości opadów w roku. Aby uzyskać średnią wartość opadów dla kilku lat, trzeba by było wykonać kilka zapytań i podać w nich różne wartości lat.

Byłoby o wiele wygodniej, gdyby można to było zrobić za pomocą języka SQL. Klauzula `GROUP BY` umożliwia grupowanie wartości w określonej kolumnie przed wykonaniem funkcji agregacji dla każdej z grup:

```
SELECT YEAR(date), AVG(daily_precip)
    FROM DailyWeatherReports
    GROUP BY YEAR(date);
```

Klauzula `GROUP BY` razem ze skalarną funkcją `YEAR` umożliwia pogrupowanie tabeli według wspólnych wartości roku, a następnie wykonanie funkcji `AVG` dla wszystkich wierszy z poszczególnych grup. Zapytanie można jeszcze bardziej uściślić za pomocą klauzuli `HAVING`, która jest nieco podobna do klauzuli `WHERE`, ale dotyczy warunków ograniczających dla podklauzuli `GROUP BY`. Przykładowo, aby uzyskać listę lat, w których średnia wartość opadów przekroczyła 50 mm, można wykorzystać następujące zapytanie:

```
SELECT YEAR(date), AVG(daily_precip)
    FROM DailyWeatherReports
    GROUP BY YEAR(date)
    HAVING AVG(daily_precip) > 50;
```

Dzięki możliwości łączenia powyższych własności można tworzyć niezwykle skomplikowane zapytania. Na przykład, aby uzyskać średnią wartość opadów dla lat 1990 – 2000 dla wszystkich miast o rocznej średniej wartości opadów przekraczającej 50 mm, można wykorzystać następujące zapytanie:

```
SELECT YEAR(date), AVG(daily_precip)
    FROM DailyWeatherReports
    WHERE YEAR(date) BETWEEN 1990 AND 2000
    GROUP BY YEAR(date)
    HAVING AVG(daily_precip) > 50;
```

Modyfikowanie schematu tabel

W pewnych sytuacjach trzeba dodać bądź usunąć kolumnę z tabeli. Można to zrobić za pomocą instrukcji `ALTER TABLE`, która ma wiele możliwych zastosowań. W tym podrozdziale zademonstruję najczęściej stosowane:

```
ALTER TABLE NazwaTabeli
    ADD NazwaKolumny TypKolumny atrybuty...;

ALTER TABLE NazwaTabeli
    DROP COLUMN NazwaKolumny;

ALTER TABLE NazwaTabeli
    CHANGE COLUMN NazwaKolumny Nowe_szczegóły;

ALTER TABLE NazwaTabeli
    RENAME AS NowaNazwaTabeli;
```

Modyfikowanie schematu tabeli jest operacją, której nie należy wykonywać często. Trzeba mieć pewność, że wykonywane działania są prawidłowe. Celem projektowania jest utworzenie struktury bazy danych, która jest wydajna, elastyczna, skalowalna i zapewni spełnienie potrzeb użytkowników w przyszłości. Jeśli zbyt często musimy modyfikować schematy tabel, może to oznaczać konieczność powtórnego przeprowadzenia procesu projektowania tabel. Poza tym, modyfikowanie schematu tabel jest operacją kosztowną. W niektórych systemach baz danych usunięcie kolumny wymaga dużej ilości miejsca na dysku. Często tabela jest zablokowana przez cały czas usuwania niepotrzebnych danych. Podobne, mniej lub bardziej przejściowe problemy z wydajnością, mogą się zdarzyć w przypadku dodawania nowych kolumn.

Aby dodać kolumnę, należy skorzystać z klauzuli `ADD` i wprowadzić nazwę nowej kolumny, jej typ danych oraz inne atrybuty. Na przykład, aby w tabeli `Users` dodać pole `password` służące do zapisywania haseł, można wykorzystać następujące zapytanie:

```
ALTER TABLE Users
    ADD password VARCHAR(50) NOT NULL;
```

Powyższe zapytanie dodaje nową kolumnę tekstową do tabeli Users z atrybutem NOT NULL wykluczającym użycie NULL jako wartości w dowolnym z wierszy. Dla istniejących wierszy w tabeli kolumna password zostanie ustawiona na pusty ciąg znaków ''. W języku SQL występują dwa opcjonalne słowa kluczowe — FIRST i AFTER, pozwalające na określenie miejsca w tabeli, w którym mają zostać umieszczone nowe kolumny:

```
ALTER TABLE Users
    ADD password VARCHAR(50) NOT NULL
    AFTER user_name;
```

Aby usunąć kolumnę z tabeli, należy skorzystać z klauzuli DROP COLUMN. Na przykład, aby usunąć kolumnę password, którą dodaliśmy przed chwilą, można wykorzystać następujące zapytanie:

```
ALTER TABLE Users
    DROP COLUMN password;
```

Podobnie jak inne operacje, które powodują usuwanie informacji z bazy danych, usunięcie kolumny jest trwałe i nie można go cofnąć. Dlatego właśnie takie polecenia muszą być używane ze szczególną ostrożnością (i raczej nie należy zezwalać na ich wykonywanie zwykłym użytkownikom bazy danych).

Do zmiany definicji kolumny służy klauzula CHANGE (klauzulą równoważną jest MODIFY). W przypadku wprowadzenia nazwy w nowej definicji kolumny, kolumna zostanie przemianowana. Klauzulę CHANGE można również wykorzystać do zmiany typu oraz atrybutów kolumny. Przykładowo, aby zmienić typ pola user_name w tabeli Users na ciąg 100-znakowy, można wykorzystać następujące zapytanie:

```
ALTER TABLE Users
    CHANGE COLUMN user_name VARCHAR(100) NOT NULL;
```

Aby zmienić nazwę tabeli User, można skorzystać z klauzuli RENAME:

```
ALTER TABLE Users
    RENAME AS MessageBoardUsers;
```

Instrukcję ALTER TABLE mogą wykonywać tylko użytkownicy posiadający uprawnienie ALTER.

Podsumowanie

Po lekturze tego rozdziału Czytelnik rozszerzył swoją wiedzę na temat języka SQL. Wykorzystaliśmy go do wykonywania coraz bardziej skomplikowanych zapytań i operacji na danych. Dzięki zastosowaniu transakcji można wykonać wiele instrukcji SQL w formie niepodzielnej operacji. Podczas obsługi transakcji można stosować różne poziomy izolacji i blokowania. Po lekturze tego rozdziału Czytelnik powinien wiedzieć, w jaki sposób można kwalifikować i sortować dane, wykorzystywać funkcje, a także modyfikować schemat tabel (pamiętając o kosztach tej operacji).

Po lekturze ostatnich czterech rozdziałów dysponujemy wystarczającą wiedzą na temat baz danych, aby zacząć z nich korzystać z poziomu kodu PHP. W następnym rozdziale przedstawię sposoby nawiązywania połączenia z serwerem, wykonywania zapytań i przeglądania wyników z poziomu naszej aplikacji internetowej.