Model relacyjny bazy danych : Strukturalny język zapytań (SQL) : System MySQL

Wprowadzenie do systemu bazy danych MySQL

wydanie drugie rozszerzone i poprawione

Paweł Szołtysek
pawel.szoltysek@student.pwr.wroc.pl

Spis treści:

Wrocław, grudzień 2007

1. Wstęp

MySQL (wymowa: maieskjuel) to bodaj najpopularniejszy (wg teamu deweloperskiego ponad 10 milionów instalacji) system obsługi danych. Jest on tworzony przez szwedzką firmę MySQL AB, w zgodzie ze (znów) najczęściej używanym, ustandaryzowanym językiem dostępu do bazy danych – SQL. Sam on, jest zdefiniowany przez standardy ANSI/ISO, które ewoluują już ponad 20 lat, a obecnie obowiązuje jego wersja z roku 2003, nazwana „SQL:2003".

Dzięki założeniom poczynionym na początku tworzenia projektu, bazy typu MySQL należą do najszybszych i najłatwiejszych w użytkowaniu. Dziś oferują bogaty zestaw funkcji, a jako część platform typu AMP – najczęściej używanymi rozwiązaniami w sieci Internet, pełniąc funkcję bazy danych.

Co jest warte podkreślenia, całe to oprogramowanie, włącznie z firmą je tworzącą, jest otwarte. Oparte na licencji GPL, pozwala nie tylko za darmo używać, ale i dowolnie je modyfikować pod swoje potrzeby. Dzięki temu, w zasadzie na każdej możliwej platformie systemowej czy dowolnej architekturze procesora można go instalować, i używać. Co ciekawe, w oryginale, był on zaprojektowany na system Solaris.

System został opublikowany 23 maja 1995 roku, a najnowsza jego wersja 5.1 jest z listopada 2005.

Nazwa wcale nie narodziła się – jak by się zdawało – od skrótu „mój strukturalny język zapytań", a od imienia córki jednego z twórców – „My".

W MySQL występują różne rodzaje tablic – MyISAM, Falcon, Merge, Memory (heap), Federated, Archive, Archive, CSV, Blackhole, Cluster, Brb, Example, InnoDB, solidDB, NitroEDB, BrightHouse, memcached, httpd, PBXT I inne. Domyślnie wykorzystywane są tabele typu MyISAM. Typ ten nie obsługuje transakcji, bazuje na starym kodzie, przy wielu rozszerzeniach. Ostatnio jest wypierany przez InnoDB, który to generalnie jest lepszy; nie posiada jedynie wyszukiwania pełnotekstowego.

2. Oprogramowanie

Samą w sobie bazę danych ściągamy ze strony http://www.mysql.com – do naszych niekomercyjnych celów wystarczy nam bezpłatna, nie ograniczona opcja „community" (na dzień 30-10-2007 dokładny link do MySQL w wersji 5.0 to http://dev.mysql.com/downloads/mysql/5.0.html). Możemy też skorzystać z – niestety już nie uaktualnianego, jednak wszystko mającego pakietu „Krasnal Serv". Najnowsza dostępna wersja na stronie głównej projektu (http://www.olesno.pl/~pablo/krasnal/), oznaczona jako 2.7, zawiera stosunkowo dużo oprogramowania (od serwera http Apache przez bazę MySQL wspomaganą przez phpMyAdmin po WebAlizera), to jednak posiada leciwe wersje oprogramowania (np. MySQL jest w wersji 3.23.58), przez co nie jest on przeze mnie w tym miejscu już polecany, chociaż jego instalacja jest bodaj najprostsza.

Standardowo, do silnika bazodanowego jest dołączony prosty menedżer, za pomocą którego używając składni MySQL możemy zarządzać bazą danych. Zamiast niego, można jednak z powodzeniem używać dowolnych innych. Prym tutaj wiedzie, znów darmowy, menedżer który

wykorzystuje przeglądarkę webową – phpMyAdmin. Jest on wzorem dla wielu innych tego typu programów zarządzających.

Osobiście przypadł mi jednak do gustu program SQLyog, który firmuje indyjska firma Webyog (http://www.webyog.com/). Umożliwia on pracę z bazą danych zarówno w trybie tekstowym, jak i – znacznie wygodniej, gdy do czynienia mamy z dużą bazą danych – GUI. Jako ciekawostka – jego tworzenie zostało zapoczątkowane jako praca zaliczeniowa na uniwersytet jednego z liderów projektu.

3. Składnia MySQL

W systemie MySQL, instrukcje mogą zawierać kilka linijek zapytań. Z tego powodu, po każdej linijce należy używać znaku średnika, co zwiększa też czytelność kodu. W wypadku wysyłania pojedynczego zapytania średnik można ominąć, jednak generalnie rzecz biorąc nie jest to zalecane.

Następna rzecz to komendy SQL. Ich formatowanie może być dowolne, ja jednak będę używał dużych liter, aby zwiększyć czytelność i przejrzystość tekstu.

Same komendy SQL mają to do siebie, że często mogą służyć do wielu różnych rzeczy – mam tu na myśli nie tylko to, że za pomocą komendy CREATE można tworzyć zarówno tabele jak i bazy danych – ale na przykład SELECT może nam posłużyć do… wyświetlenia aktualnej daty.

Bardzo ważne jest odpowiednie używanie znaków ` oraz ‘. Ogólnie napisać można, że tam, gdzie mamy na myśli element bazy danych – tabele, rekord czy nazwę atrybutu – użyjemy `, natomiast przy wartościach – ‘. Dopuszczalne jest ich nie używanie, jednak ich dodanie powoduje wzrost czytelności kodu, przez co jest to zdecydowanie wskazane.

4. Zarządzanie bazą danych i tabelami

W systemie MySQL możemy mieć naraz wiele baz danych. Aby je zobaczyć, używamy komendy

```
SHOW DATABASES;
```

Polecenie to wyświetli nam na naszym ekranie wszystkie bazy danych jakie mamy w naszym systemie. Aby utworzyć nową bazę danych, piszemy

```
CREATE DATABASE `nasza`;
```

Została utworzona baza danych nasza_baza. Użyj teraz polecenia SHOW, aby zobaczyć obecnie wszystkie bazy danych.

No dobrze, a jak usuwamy?

```
DROP DATABASE `nasza`;
```

Tym samym usunęliśmy przed chwilą utworzoną bazę danych.
UWAGA: To polecenie usuwa bazę danych łącznie ze wszystkimi danymi!
No dobrze. Załóżmy jeszcze raz, z powrotem bazę danych `nasza`.

```
CREATE DATABASE `nasza`;
```

Super. Aby jednak wydać konkretne polecenia MySQL, interpreter musi wiedzieć o którą bazę nam chodzi. W jaki sposób więc można pracować na tej bazie danych?

```
USE `nasza`;
```

Teraz możemy wykonywać zapytania na tej bazie danych. Aby zobaczyć obecnie istniejące tabele, wykorzystamy podobne polecenie do wyświetlania baz danych:

```
SHOW TABLES;
```

Powinien pojawić się nam komunikat informujący, iż nasza baza jest pusta – nie utworzyliśmy żadnych tabel w niej do tej pory. Jak już pewnie się domyślacie, tabele się tworzy podobnie jak bazy danych.

```
CREATE TABLE `Komputery` (`id` int NOT NULL AUTO_INCREMENT,
`Procesor` CHAR(5), `Ram` CHAR(4) DEFAULT 'DDR2',
`WielkoscRamu` int, `Wysokosc` DECIMAL(3,2), `MaDVD` BOOLEAN,
PRIMARY KEY(id));
```

Ojej. Mam nadzieję, że się nie zgubiliście. Faktycznie, w tym miejscu muszę się Wam rozgrzeszyć, co wyżej popełniłem. Otóż, stworzyłem tabelę `Komputery`. W tabeli tej mam 6 pól. Pierwsze nosi nazwę `id`. Jest to wartość typu integer. Atrybut ten nigdy nie będzie pusty, dzięki NOT NULL. Jednocześnie, typ AUTO_INCREMENT spowoduje przy dodaniu nowego rekordu (o czym będzie niżej) zwiększenie jego wartości o jeden – a przy usunięciu rekordu nie spowoduje zmiany jego wartości. Drugie pole to `Procesor`. Jest to zwykłe pole znakowe, które przechowuje tekst o długości nie większej niż 5 znaków. Kolejne pole to `Ram`; jest ono podobne do pola `Procesor`, jednak posiada parametr DEFAULT. Oznacza to, iż przy dodaniu nowego rekordu do bazy danych i nie podaniu jego wartości, argument ten przyjmie wartość właśnie tutaj podaną – w tym przypadku będzie to 'DDR2'. Kolejnym ciekawym typem jest DECIMAL(x,y), użyty w polu `Wysokosc`. W tym miejscu, x oznacza maksymalną liczbę cyfr przed, a y – po przecinku. BOOLEAN to z kolei standardowy typ jednobitowy

prawda/fałsz. Istnieje więcej typów pól. Najważniejsze, poza wyżej wymienionymi, to BLOB (ciąg znaków ograniczony pamięcią fizyczną), DATE (data w formacie YYYY-MM-DD), i YEAR (rok). Pomijając te typy, jest jeszcze dużo innych dostępnych w MySQL, chociaż bardzo rzadko używanych. Możecie je „podejrzeć", chociażby używając dowolnego menedżera MySQL.
Jeszcze na końcu zawarliśmy informację, iż kluczem głównym jest `id`.

No dobrze. Tak dużo nabroiliśmy, a jak zobaczyć wyniki?

```
DESCRIBE `Komputery`;
```

Zapytanie to wyświetli nam strukturę tej bazy danych – czyli poda wszystkie informacje na temat pól.

Po utworzeniu tabeli możemy oczywiście dodawać do niej pola według naszego uznania. Możemy tego zrobić w następujący sposób:

```
ALTER TABLE `Komputery` ADD `HDD` int;
```

Podobnie można usunąć pola:

```
ALTER TABLE `Komputery` DROP `Ram`;
```

Aby zmienić nazwę pola, napiszemy:

```
ALTER TABLE `Komputery` CHANGE `WielkoscRamu` `RAM` int;
```

Usunęliśmy więc pole `Ram`, a w jego miejsce wstawiliśmy `WielkoscRamu`.
Możemy zmieniac także nazwy całych tabel:

```
ALTER TABLE `Komputery` RENAME `Desktopy`;
```

Tak samo zadziała

```
RENAME `Komputery` TO `Desktopy`;
```

W analogiczny sposób można tabele przenosić między bazami danych.

5. Operacje na rekordach

Mając tabelę, wypadałoby w niej przechowywać jakieś dane. Rekordy dodajemy w sposób następujący:

```
INSERT INTO `Desktopy` VALUES (‘’, ‘AMD’, ‘512’, ‘540,12’, ‘0’, ‘80’);
```

Pierwsze pole jest autonumerujące, więc jego wartości nie musieliśmy ustalać – zostanie ono uzupełnione automatycznie. Należy zauważyć, iż można dodawać też dane tylko w wybranych polach:

```
INSERT INTO `Desktopy` (`Procesor`,`Ram`) VALUES (‘intel’,‘256’);
```

Pozostałe pola przyjmą wartości domyślne, lub będą puste.

Polecenie REPLACE działa dokładnie tak samo jak INSERT z jedną różnicą – jeśli istnieje już w tabeli wiersz o tej samej wartości klucza, zostanie on podmieniony (INSERT nie wykona wtedy żadnej akcji i zgłosi ostrzeżenie).

Do dodania rekordów z pliku służy zapytanie LOAD.

```
LOAD DATA INFILE ‘C:\Moj_Folder\baza.txt’ IGNORE INTO TABLE `Desktopy`;
```

W ten sposób skopiujemy dane z pliku znajdującego się NA serwerze w katalogu jak podanym. Ignorować będziemy powtórzenia indeksu (klucza), a tabelą do której zostaną dodane rekordy będzie tabela `Desktopy`. Klient MySQL pozwala nam na przesłanie pliku z naszego komputera, i bezpośrednie wstawienie go do tabeli w sposób następujący:

```
LOAD DATA LOCAL INFILE ‘C:\Moj_Folder\baza.txt’ REPLACE INTO TABLE `Desktopy`;
```

Zmieniłem tu też jednocześnie fakt kolizji kluczy – teraz przy takim wystąpieniu rekord będzie zmieniany na nowszą „wersję”.
Po podaniu nazwy tabeli możemy dopisać opcje importowania. Będą one definiowały format, w jakim chcemy przetransportować dane z pliku do tabeli.

```
LOAD DATA LOCAL INFILE ‘C:\Moj_Folder\baza.txt’ IGNORE INTO TABLE `Desktopy` FIELDS TERMINATED BY ‘,’ ENCLOSED BY ‘”’ ESCAPED BY ‘\\’ LINES STARTING BY ‘’ TERMINATED BY ‘\n’;
```

W ten sposób zapiszemy odczyt ze standardowego pliku formatu CSV. Po zdefiniowaniu formatu możemy do zapytania dodać jeszcze klauzulę ignorowania linii – IGNORE 5 LINES sprawi że 5 pierwszych wierszy w pliku zostanie zignorowanych, a *czytanie* rozpocznie się dopiero od szóstej. Po tym, możemy jeszcze dodać listę kolumn, w kolejności w której dane mają być wpisywane do tabeli.

Chcielibyście zapewne zobaczyć teraz, jak wygląda Wasza tabela. Nic prostszego!

```
SELECT * FROM `Desktopy`;
```

I w ten sposób doszliśmy do jednej z najważniejszych komend – zapytania SELECT.

6. Pobieranie informacji z bazy danych.

W powyższym zapytaniu dzięki „*”, zostaną nam wyświetlone wszystkie pola z tabeli. Oczywiście, nie musimy za każdym razem wyświetlać wszystkich atrybutów (jest to nawet niewskazane, aby wywoływać te których w danej chwili nie potrzebujemy ze względów wydajnościowych). Aby odczytywać wybrane pola z tabeli, napisalibyśmy

```
SELECT `Procesor`,`Ram` FROM `Desktopy`;
```

Możemy także ograniczać rekordy które otrzymamy. Powinniśmy to także robić za każdym razem.

```
SELECT * FROM `Desktopy` WHERE `Ram`='512';
```

Taki wybór możemy precyzować na wiele różnych sposobów.

```
SELECT * FROM `Desktopy` WHERE `Procesor` IN ('cyrix','intel');
```

Otrzymamy tutaj listę wszystkich desktopów, które mają procesor cyrixa lub Intela (dziedzina pola Procesor to cyrix i Intel). Aby wyświetlić rekordy z wartością pomiędzy dwoma liczbami, napiszemy

```
SELECT * FROM `Desktopy` WHERE `Ram` BETWEEN '500' AND '700';
```

Zauważ, że możesz to zapytanie inaczej sformułować:

```
SELECT * FROM `Desktopy` WHERE `Ram`>'500' AND `Ram`<'700';
```

Wspomniany operator AND służy do łączenia kilku różnych warunków – możesz też użyć analogicznie operatora OR. Jednocześnie należy wspomniec, iż zamiast AND możesz użyc &&, a zamiast OR - ||.

Budując bazę, w której będziesz pozwalał użytkownikom wyszukiwać dane, na pewno docenisz następujące polecenie:

```
SELECT * FROM `Desktopy` WHERE `Procesor` LIKE 'a%';
```

W powyższej formule, wyświetli wszystkie rekordy, w których pole `Procesor` zaczyna się literką „a" i ma dowolną ilość znaków.
Wyszukiwanie może być znacznie bardziej złożone, jeśli użyjemy funkcji REGEXP, czyli wyrażeń regularnych. Aby wyświetlić desktopy, których procesory mają w nazwie tylko litery, napiszemy

```
SELECT * FROM `Desktopy` WHERE `Procesor` REGEXP '[a-z]';
```

Dostępne meta znaki:

```
.               - odpowiada dowolnemu znakowi, w tym również nowemu wierszu
^               - początek wiersza
$               - koniec wiersza
[...]           - dowolny znak ze zbioru
[^...]          - dowolny znak spoza zbioru
[:klasa:]       - definicja klasy znaków, takich jak [:digit:], [:alpha:]
*               - liczba 0 lub większa
+               - liczba 1 lub większa
?               - liczba 1 lub 0
{x}, {x,y}, {x,} - Dopasowanie x-krotne. Od x do y i co najmniej x
|               - ogranicznik wartości alternatywnych
(...)           - definicja wyrażenia składowego
```

Polecenia możesz negować:

```
SELECT * FROM `Desktopy` WHERE `Procesor` NOT IN ('amd','cyrix');
```

Raz już tego użyliśmy – gdzie?

Wybranie rekordów w ten sposób wyświetli nam je w kolejności w jakiej były one dodawane. Jednak, rekordy można bezpośrednio sortować.

```
SELECT * FROM `Desktopy` ORDER BY `Procesor` ASC;
```

Wyświetli to rekordy w kolejności alfabetycznej zgodnie z polem `Procesor`, rosnąco. Aby odwrócić wyniki, zamiast ASC należy użyć DESC. Możesz też sortować na niższych poziomach, po prostu dopisując po przecinku nazwę pola i tryb sortowania. Może się tak zdarzyć (np. z niewystarczającej

wielkości wyświetlacza) iż będziemy chcieli wyświetlić tylko cześć rekordów. Zapytanie wtedy będzie wyglądało następująco:

```
SELECT * FROM `Desktopy` LIMIT 1,10;
```

A zwrócone zostanie nam pierwszych dziesięć rekordów począwszy od pierwszego (nie zerowego!).

Jeśli chcemy otrzymać losowy rekord z bazy danych, napiszemy:

```
SELECT * FROM `Desktopy` ORDER BY RAND() LIMIT 1;
```

Nieraz może się zdarzyć tak, że wyświetlając pewien fragment bazy danych, wartości dla różnych rekordów będą identyczne – domyślnie będą się wtedy duplikowało. Aby tego uniknąć, używamy polecenia DISTINCT w następujący sposób:

```
SELECT DISTINCT `Procesory` FROM `Desktopy`;
```

W ten sposób jesteśmy pewni, iż każdy producent zostanie wyświetlony dokładnie jeden raz.

Podobnie jak w innych językach programowania, SQL obsługuje instrukcje warunkowe. Najprostszą jest funkcja IF.

```
SELECT * FROM `Desktopy` WHERE IF(`Procesor` =
'AMD',`Ram`='512',`Ram`='1024');
```

Funkcja IF oblicza wyrażenie i zwraca wartość drugiego argumentu jeśli wyrażenie jest prawdziwe, lub też trzeciego, jeśli fałszywe.
No dobrze, ale co jeśli chcemy narzędzie bardziej rozbudowane?
Do tego służy funkcja CASE. Przykład użycia:

```
SELECT `nick`, CASE WHEN `nick`='Jola' THEN 'Wcale nie
lubie tego imienia' WHEN `nick`='Zbigniew' THEN 'Zbyszek
jest fajny' ELSE 'bez komentarza' END FROM `Uzytkownicy`;
```

Polecenie SELECT może nam również dostarczyć innych danych. Aby poznać największy numer id, napiszemy:

```
SELECT MAX(id) FROM `Desktopy`;
```

W miejsce MAX, możemy wpisac MIN (dla minimalnej wartości), COUNT (dla ilości rekordów dla konkretnego zapytania), SUM (dla sumy pól) czy AVG (dla średniej wielkości z pól). Wypróbuj też (już bez pola w nawiasie) USER() (wyświetli aktualnego użytkownika i nazwę komputera z którego się łaczy) i NOW() (wyświetli aktualną datę i godzinę serwera). Można je oczywiście łączyć z wyrazami kluczowymi takimi jak WHERE, ORDER BY czy GROUP BY.

Podobnie, za pomocą zapytania SELECT możemy od razu wykonywać nawet nieco bardziej złożone działania, matematyczne:

```
SELECT SQRT(16%5);
```

ale także, oczywiście, z zawartością kolumn tabeli. Typowym przykładem tu jest konkatenacja.

```
SELECT CONCAT(`Procesor`,' ',`RAM`) FROM `Desktopy`;
```

Powyższe wyrażenie po prostu połączy wartości pól `Procesor` i `RAM`, rozdzielając je pojedynczą spacją. Do *ładnego* wyświetlenia w tym wypadku nazwy przydaje się przełącznik AS:

```
SELECT CONCAT(`MaDVD`,' ',`HDD`) AS 'Storage' FROM
`Desktopy`;
```

Wyniki możemy przesyłać do pliku. Robimy to w sposób następujący:

```
SELECT * INTO OUTFILE 'C:\baza.txt' FROM `Desktopy`;
```

Przed wyrazem FROM możemy dodać przełączniki podobnie jak w zapytaniu LOAD.

7. Operacje na rekordach – ciąg dalszy

Wróćmy do operowania na rekordach w ogóle. Dowiemy się teraz jak możemy usunąć konkretny rekord.

```
DELETE FROM `Desktopy` WHERE `Procesor`= 'intel';
```

UWAGA: Powyższe zapytanie usunie nam WSZYSTKIE rekordy, które w polu `Procesor` mają wartość 'intel'.
Przy usuwaniu możemy używać w podobnym zakresie co przy poleceniu SELECT wszelkich przełączników, jak na przykład IN czy LIKE. Jest zalecanym, aby każdorazowo dodawać polecenie LIMIT, aby przez przypadek nie stracić danych których nie chcieliśmy usunąć.

W naszej bazie danych możemy również zmieniać wartości pól po utworzeniu rekordów.

```
UPDATE `Desktopy` SET `Ram`='1024' WHERE `Procesor`='AMD';
```

UWAGA: Ponownie, powyższe zapytanie wpłynie na WSZYSTKIE rekordy, które posiadają wartość 'AMD' w polu `Procesor`.
Oczywiście w podobny sposób możemy dokonywać zmian wielu pól naraz:

```
UPDATE `Desktopy` SET `Ram`='512',`HDD`='160' WHERE
`Procesor`='AMD';
```

8. Wartość bez wartości

Jest jedna wartość w MySQL której znaczenia nie można wprost określić.
Spróbujmy wstawić pusty rekord do bazy danych.

```
INSERT INTO `Desktopy` VALUES ();
```

Spodziewamy się, że zapytanie to dodało nam pusty rekord do bazy danych. Spróbujmy więc odczytać ten rekord.

```
SELECT `Procesor` FROM `Desktopy` WHERE `Procesor`='';
```

Brak rekordów. Co się stało? Przecież przed chwilą dodaliśmy rekord, który nie miał żadnej wartości.

```
SELECT `Procesor` FROM `Desktopy` WHERE `Procesor` IS NULL;
```

Teraz widzimy fragment rekordu który dodaliśmy. No dobrze, jaką więc wartość dodaliśmy do bazy w tym miejscu? Tego określić nie można. Jak to nie?

```
SELECT NULL!=0, NULL=0;
```

Co więcej, nie można nawet porównać jej z samą sobą (!):

```
SELECT NULL!=NULL, NULL=NULL;
```

Wszystkie te zapytania zwrócą nam wynik NULL.
Jeśli będziemy używać polecenia SELECT, wartość NULL będzie wyświetlona zawsze na początku, niezależnie od sortowania które użyliśmy. Można jednak użyć polecenia IF po ORDER BY, aby ustalić, czy ma ona zostać wyświetlona na początku czy na końcu.

```
SELECT * FROM `Desktopy` ORDER BY IF(`Procesor` IS
NULL,1,0), `Procesor`;
```

Powyższe zapytanie wyświetli nam tabelę `Desktopy` posortowaną rosnąco w polu `Procesor`, a jednocześnie wszystkie wartości NULL w tym polu zostaną wyświetlone na końcu.

9. Praca na dwóch tabelach jednocześnie

Siła relacyjnych baz danych tkwi w zdolności tworzenia relacji między jednym elementem a drugim. Pozwala to łączyć informacje z wielu tabel w łatwy sposób.

Na początek zdefiniujmy sobie tabele jak poniżej:

```
CREATE TABLE `Uzytkownicy` (`id` INT NOT NULL AUTO_INCREMENT,
`nick` CHAR(20), `haslo` char(20), PRIMARY KEY (id));
CREATE TABLE `Wiadomosci` (`id` INT NOT NULL AUTO_INCREMENT,
`id_nadawcy` INT, `wiadomosc` char(20), PRIMARY KEY (id));
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Marek',
'tajnehaslo');
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Jola',
'abc');
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Franek',
'abc');
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Zbigniew',
'innehaslo');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(1,
'Jestem wesoly Marek');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(2,
'Spiewam sobie');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(3,
'Mam trudne haslo');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(4,
'Mam inne haslo');
```

Mamy dwie tabele - `Uzytkownicy` oraz `Wiadomosci`. Nauczymy teraz się odpowiednio wykorzystywać polecenie SELECT. Na początek przełącznik UNION.

```
SELECT `haslo` FROM `Uzytkownicy` UNION SELECT `wiadomosc`
FROM `Wiadomosci`;
```

Powyższe zapytanie zwróci nam jedną kolumnę, z wartościami z pola `haslo` oraz `wiadomosc`. UNION pozbywa się powtarzających wartości. Aby temu zapobiec, dopisujemy po UNION słowo ALL. UWAGA: Należy pamiętać, iż UNION działa tylko z zapytaniem SELECT. UNION jest także często wykorzystywany przy SQL INJECTION (co to jest, będzie opisane dalej).

Pełne łączenie tabel dokonuje się następująco:

```
SELECT `Uzytkownicy`.*, `Wiadomosci`.* FROM
`Uzytkownicy`,`Wiadomosci`;
```

Tabele łączymy ekwiwalentnie w sposób następujący:

```
SELECT `Uzytkownicy`.`nick`, `Wiadomosci`.`wiadomosc` FROM
`Uzytkownicy` INNER JOIN `Wiadomosci` ON `Uzytkownicy`.`id`
= `Wiadomosci`.`id_nadawcy`;
```

co ciekawe, taki sam wynik otrzymamy w przypadku może bardziej intuicyjnego zapytania

```
SELECT `Uzytkownicy`.`nick`, `Wiadomosci`.`wiadomosc` FROM
`Uzytkownicy`,`Wiadomosci` WHERE `Uzytkownicy`.`id` =
`Wiadomosci`.`id_nadawcy`;
```

Aby dokonać złączenia lewo- lub prawostronnego, zamieniamy INNER z odpowiednio LEFT lub RIGHT.

Gdybyśmy zmienili nazwę pola w tabeli `Uzytkownicy` z `id` na `id_nadawcy`, moglibyśmy napisać

```
SELECT `Uzytkownicy`.`nick`, `Wiadomosci`.`wiadomosc` FROM
`Uzytkownicy` LEFT JOIN `Wiadomosci` USING `id_nadawcy`;
```

Byłoby to wtedy równoważne co do wyniku z poprzednim zapytaniem.
Jest też opcja złączenia naturalnego.

```
SELECT * FROM `Uzytkownicy` NATURAL LEFT JOIN `Wiadomosci`;
```

Jest to równoznaczne z poprzednim zapytaniem, jeśli tylko po słowie kluczowym USING wpisalibyśmy nazwy wszystkich kolumn.

Podczas zapytania SELECT, używając innych tabel także możemy zawężać wyniki:

```
SELECT * FROM `Wiadomosci` WHERE `id_nadawcy` IN (SELECT
`id` FROM `Uzytkownicy` WHERE `nick` = 'Marek');
```

Przykład: Wyświetl sumę długości wiadomości każdej osoby.

```
SELECT `Uzytkownicy`.`nick`,
SUM(LENGTH(`Wiadomosci`.`wiadomosc`)) AS 'Dlugosc
wiadomosci' FROM `Uzytkownicy` JOIN `Wiadomosci` ON
(`Wiadomosci`.`id_nadawcy`=`Uzytkownicy`.`id`) GROUP BY
`Uzytkownicy`.`id`;
```

10. Transakcje

Transakcja jest zbiorem instrukcji, które tworzą jedną nierozerwalną całość, i nie mogą być wykonane oddzielnie, ponieważ stracą wtedy swoją właściwość. Jest to więc ważna rzecz, szczególnie jeśli mowa o bazach danych. W MySQL można transakcje wykonać na kilka sposobów.
Instrukcje wchodzące w skład transakcji można na przykład zgrupować, a następnie otoczyć zapytaniami LOCK oraz UNLOCK.

```
LOCK TABLE `Uzytkownicy` WRITE;
```

Takie zapytanie zablokuje dla wszystkich klientów serwera MySQL możliwość tak czytania z, jak i zapisywania do tabeli `Uzytkownicy`. Zamiast WRITE można użyć READ, będziemy wtedy mieli możliwość tylko pobierania danych z tabeli, przy jednoczesnym zablokowaniu dostępu dla każdego innego klienta. Blokadę zdejmujemy za pomocą

```
UNLOCK TABLES;
```

Możemy też korzystać przy zapytaniach UPDATE z aktualizacji względnej, to znaczy używać danych nie tych, które pobraliśmy wcześniej, a z konkretnych danych które są w chwili wykonywania zapytania w bazie danych. Takie rozwiązania zachowują jednak tylko część właściwości transakcji – bowiem w wypadku na przykład przerwy w dostawie prądu, zostanie wykonana tylko jej część, a to przeczy definicji transakcji.
MySQL oferuje narzędzie, znacznie wygodniejsze, za którego pomocą można przeprowadzać transakcje.
W MySQL domyślnie włączone jest auto-zatwierdzanie zapytań. Jeśli udałoby się nam je wyłączyć, wtedy moglibyśmy napisać kilka zapytań naraz, a dopiero potem je zatwierdzić. Jeśli tylko używamy typu tabeli, który obsługuje transakcje, możemy napisać

```
START TRANSACTION;
```

Wszystkie zapytania, które wpiszemy od tego miejsca zostaną zachowane w pamięci, nie będą jednak wykonane aż do momentu wpisania

```
COMMIT;
```

Między tymi dwoma zapytaniami możemy wpisać dowolną ilość zapytań, które wchodzą w skład transakcji. Zostaną one wykonane jednocześnie po wpisaniu COMMIT. Instrukcja START TRANSACTION wyłącza więc tryb automatycznego zatwierdzania.
Jeśli podczas wpisywania transakcji na bazie danych zorientujemy się, że popełniliśmy błąd, możemy łatwo cofnąć całą transakcję, wpisując zamiast COMMIT polecenie ROLLBACK.
UWAGA: Zamiennie z START TRANSACTION można użyć BEGIN.

Jeśli chcemy, możemy tymczasowo wyłączyć auto-zatwierdzanie zapytań. Zrobimy to za pomocą polecenia

```
SET AUTOCOMMIT = 0;
```

Od tego momentu wszystkie zapytania które wyślemy do bazy danych nie będą wykonane dopóki nie wywołamy COMMIT. Ustawiając wartość na 1, włączymy powrotem auto-zatwierdzanie.
UWAGA: Istnieją zapytania, które automatyczne wykonują COMMIT. Jest to związane ze strukturą języka MySQL. Do tych zapytań należy m.in. BEGIN, CREATE TABLE, DROP TABLE, LOCK TABLE, UNLOCK TABLES itp.

Przydatnym poleceniem jest

```
SAVEPOINT identyfikator;
```

Wyrażenie to ustala miejsce w transakcji, do którego później można wrócić przez wyrażenie

```
ROLLBACK TO SAVEPOINT identyfikator;
```

Po napisaniu tego zapytanie wszystkie zmiany które uczyniliśmy od miejsca SAVEPOINT będą usunięte.

11. Klucze obce

Klucz obcy pozwala stwierdzić, że indeks jednej tabeli jest powiązany z wartością w polu innej. Wymusza to więzy integralności, sprawia także, że nie można wtedy użyć nieodpowiedniej wartości. Weźmy pod uwagę tabele, które utworzyliśmy kilka stron wcześniej:

```
CREATE TABLE `Uzytkownicy` (`id` INT NOT NULL AUTO_INCREMENT,
`nick` CHAR(20), `haslo` char(20), PRIMARY KEY (id));
CREATE TABLE `Wiadomosci` (`id` INT NOT NULL AUTO_INCREMENT,
`id_nadawcy` INT, `wiadomosc` char(20), PRIMARY KEY (id));
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Marek',
'tajnehaslo');
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Jola',
'abc');
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Franek',
'abc');
INSERT INTO `Uzytkownicy`(`nick`, `haslo`) VALUES('Zbigniew',
'innehaslo');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(1,
'Jestem wesoly Marek');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(2,
'Spiewam sobie');
INSERT INTO `Wiadomosci`(`id_nadawcy`, `wiadomosc`) VALUES(3,
'Mam trudne haslo');
```

Widać, że tabela `Wiadomosci` zawiera pole `id_nadawcy`, które służy do powiązania rekordów zawierających treść wiadomości z ich autorami. Użycie klucza obcego zapobiegnie wstawieniu nieprawidłowego id użytkownika do tabeli `Wiadomosci`, a także dzięki niemu można nałożyć takie ograniczenie, że po usunięciu użytkownika zostaną usunięte też jego wiadomości. Kluczem obcym określamy pole w tabeli potomnej (tutaj: `Wiadomosci`), dla pola w tabeli nadrzędnej (tutaj: `Uzytkownicy`). Podczas tworzenia tabeli `Wiadomosci` napiszemy więc:

```
CREATE TABLE `Wiadomosci` (`id` INT NOT NULL
AUTO_INCREMENT, `id_nadawcy` INT, `wiadomosc` char(20),
PRIMARY KEY (id), FOREIGN KEY (id_nadawcy) REFERENCES
`Uzytkownicy` (`id`) ON DELETE CASCADE);
```

Możemy też nakazać bazie danych nie usuwać rekordów bezpośrednio po usuwaniu użytkownika. Przy ustawieniu ON DELETE SET NULL, pola gdzie wcześniej było ID użytkownika przyjmą wartość NULL. Gdy pracujemy na tabeli która nie obsługuje kluczy obcych, musimy radzić sobie sami. Nie jest to jednak zbyt skomplikowane. Przy wprowadzaniu danych musimy po prostu uważać, by nie wpisać niedozwolonej wartości w polu `id_nadawcy`, a usuwanie załatwiamy w sposób następujący:

```
DELETE `Uzytkownicy`, `Wiadomosci` FROM `Uzytkownicy` LEFT
JOIN `Wiadomosci` USING (Uzytkownicy.id) WHERE
`Uzytkownicy`.`id`=5;
```