MySQL

MySQL to najpopularniejszy na świecie system zarządzania bazą danych typu Open Source. Na stronie http://www.mysql.com są dostępne wszystkie najświeższe informacje na temat tego systemu.

1 Instalacja MySQL

MySQL może być zainstalowany zarówno na systemach Linuxowych jak i Windowsowych. Zalecane jest używanie MySQL na systemie Linux. Poniższe kroki ukazują sposób instalacji MySQL na systemie Linux (instalacja na systemie Windows jest intuicyjna i nie powinna nastręczyć problemów):

1. `cd /usr/local/src /mysql-x.xx-.xx`
   Włącza katalog MySQL na górny poziom hierarchii katalogów
2. `./configure –prefix=/usr/local/mysql`
   Uruchamia program konfiguracji, MySQL zostanie zainstalowany w katalogu usr/local/mysql
3. `make`
   Kompiluje MySQL
4. `make install`
   Instaluje MySQL
5. `echo “/usr/local/mysql/lib/mysql” >> /etc/ld.so.conf`
   Dodaje do pliku konfiguracji polecenia ldconfig bibliotekę MySQL. Katalogi w tym pliku konfiguracji są przeszukiwane pod kątem plików bibliotek podczas uruchamiania system Linux albo podczas wykonywania polecenia `ldconfig`.
6. ldconfig –v | grep libmysqlclient
   Polecenie `ldconfig` odczytuje katalogi wymienione w `/etc/ld.so.conf` i ładuje do pamięci podręcznej wszystkie znalezione biblioteki (grep użyty do ograniczenia wyświetlania).
7. `echo „usr/local/mysql/bin/safe_mysqld > /dev/null &” >> /etc/rc.d/rc.local`
   Dodaje polecenia uruchamiania MySQL do pliku `/etc/rc.d/rc.local` po to by MySQL był automatycznie uruchamiany podczas startu systemu.
8. ./scripts/my_sql_install_db
   Inicjuje bazę danych.
9. `usr/local/mysql/bin/safe_mysqld > /dev/null`
   Uruchamia serwer MySQL.

Można teraz przetestować instalację korzystając np. z programu narzędziowego mysqlshow, lub mysql. Po połączeniu z serwerem np.:

```
mysql –h host –u user –p
```

Można zacząć wpisywać polecenia, np.: `show databases` – wyświetlone zostaną dostępne bazy danych. W celu korzystania z jednej z nich należy posłużyć się poleceniem `use` np. `use test`, a następnie wpisywać komendy SQL zakończone średnikiem.

2 Łączenie się z bazą danych

W celu połączenia się z bazą danych należy na ogół dostarczyć nazwy hosta (jeśli serwer mysql jest na innym komputerze niż ten, na którym pracujemy), nazwy użytkownika oraz hasła.

```
shell> mysql –h nazwa_hosta –u nazwa_uzytkownika –p
Enter password: ************
```

Jeśli podaliśmy poprawne hasło powinniśmy zobaczyć:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 3.22.20a-
log
Type 'help' for help.
mysql>
```

Niektóre instalacje mysql pozwalają połączyć się jako user anonymous na hoście lokalnym: wystarczy wtedy w shellu wpisać `mysql`.

W celu rozłączenia z bazą i zakończenia pracy wpisujemy:

```
mysql> quit
Bye
```

Można się również rozłączyć wpisując wciskając Ctrl + ‘D’.

3 Wprowadzanie zapytań

Jeżeli jesteś już połączony możesz wprowadzać komendy SQL (komendy kończ znakiem średnika). Prosty przykład komendy, która zwróci nr wersji serwera oraz aktualna datę:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| version()    | CURRENT_DATE |
+--------------+--------------+
| 4.0.16a-log  | 2003-11-29   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

Wielkość liter w poleceniach nie ma znaczenia, poniższe komendy są zatem sobie równoważne:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Kolejny przykład pokazuje korzystanie z mysql'a jak z kalkulatora:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

Uwaga: możliwe jest wprowadzanie kilku komend w jednej lini (wystarczy oddzielić je średnikami) jak i wprowadzenie jednej długiej komendy w kilku liniach.

Przykład komendy składającej się z kilku lini:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
```

Prompt	Znaczenie
`mysql>`	Gotowość do wprowadzani komendy
`->`	Oczekiwanie na wprowadzenie kolejnej lini komendy składającej się z wielu lini.
`'>`	Oczekiwanie na wprowadzenie kolejnej lini, zawierającej string zaczynający się od znaku ‘'’
`">`	Oczekiwanie na wprowadzenie kolejnej lini, zawierającej string zaczynający się od znaku ‘"’

3.1 Tworzenie bazy danych

W celu korzystania bazy danych, która ma na celu przechowywać jakieś dane należy zrobić kilka rzeczy:

- Utworzyć bazę danych
- Utworzyć jedną lub wiele tabel
- Załadować dane do tabeli/tabel (np. załadować je z pliku)
- Wykorzystywać dane zawarte w tabelach na różne sposoby

Komenda SHOW pozwala sprawdzić jakie dane obecnie istnieją na serwerze:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

Baza danych mysql istnieje zawsze i są w niej przechowywane uprawnienia użytkowników.

Utworzenie bazy danych:

CREATE DATABASE Test;

Wybranie jej jako bazy, na której pracujemy:

```
Use Test;
```

Utworzenie tabeli bazy danych:

```
CREATE TABLE Pracownicy (
     ID INT NOT NULL AUTO_INCREMENT,
     Imie CHAR(20) NOT NULL,
     Nazwisko CHAR(30) NOT NULL,
     Stanowisko CHAR(20) NOT NULL,
     PRIMARY KEY (ID)
);
```

Dodanie przykładowego rekordu:

```
INSERT INTO Pracownicy(Imie, Nazwisko, Stanowisko)
VALUES („Paweł”, „Skrzyński”, „wykładowca”)
```

Aby wyświetlić wszystkie tabele znajdujące się w aktualnej bazie danych wykonujemy polecenie:

```
     SHOW TABLES;
```

Aby wyświetlić informację o strukturze tabeli wykonujemy komendę:

```
     DESCRIBE PRACOWNICY
```

3.2 Tworzenie skryptów

Jeżeli tworzymy bazę danych składającą się z kilku lub więcej tabel wygodnie jest przygotować sobie skrypt z komendami SQL w osobnym pliku tekstowym. Skrytpy są wygodne nie tylko podczas tworzenia bazy danych. Wywołując mysql podajemy jako parametr nazwę tego skryptu:

```
Shell> mysql <  nazwa_pliku
```

Lub jeśli potrzebujemy specyfikować parametry połączenia:

```
Shell> mysql –h host –u user –p <  nazwa_pliku
Enter password: ********
```

Jeżeli używamy systemy Windows i mamy w pliku jakieś specjalne znaki, które mogą sprawić problem dobrze się posłużyć opcją –e:

```
dos> mysql –e „nazwa_pliku”
```

Dlaczego warto używać skryptów:

- Jeżeli jakieś zapytanie wykonujemy często pozwala nam to uniknąć wpisywania go za każdym razem (czasami zapytania bywają długie ☺).
- Możemy szybko tworzyć zapytania podobne do istniejących poprzez kopiowanie odpowiednich fragmentów.
- Jeśli zrobimy jeden błąd to nie musimy przepisywać wszystkiego.
- Jeśli skrypt generuje „duże wyjście” to możemy je przekierować do pliku:
  `shell> mysql < batch-file > mysql.out`
- Istnieje wiele edytorów wspomagających tworzenie zapytań w SQL (kolorowanie składni itp.)

Skryptów możemy używać również pracując w mysql:

```
mysql> source nazwa_pliku;
```

3.3 Podstawowe typy danych w MySQL

3.3.1 Typy danych, INT

INT(ile) [Unsigned]

Ten typ danych przechowuje liczby z zakresu od -2147483648 do 2147483647. Użycie opcjonalnego pola Unsigned oznacza, iż nie chcemy przechowywać informacji o ewentualnym znaku liczby (np. dane są zawsze dodatnie) - wtedy zakres pola INT zmienia się na : 0 do 4294967295. Przykład:

```
wiek INT unsigned;
kwota_na_koncie INT;
```

3.3.2 Typy danych, FLOAT

FLOAT

Liczba zmiennoprzecinkowa pojedynczej precyzji (domyślnie MySQL używa 4 bajtów do przechowywania tej informacji). Jako rozszerzenie standardu ANSI dopuszcza specyfikację precyzji.

Przykład:

```
cena  FLOAT;
```

3.3.3 Typy danych, DOUBLE/REAL

Liczba o powiększonej precyzji. Nie dopuszcza na specyfikację precyzji. Dkładnośc typu REAL jest mniejsza niż DOUBLE.

Przykład:

```
liczbad     DOUBLE;
liczbar     REAL;
```

3.3.4 Typy danych, DATE

Pole tego typu przechowuje dane związane z datą. Domyślnym formatem przechowywania danych jest "RRRR-MM-DD"!!! Zakres tego pola sięga od "0000-00-00" do "9999-12-31". MySQL dostarcza programistom wiele poleceń związanych z manipulacją danymi związanymi z datami. Więcej informacji można wyszukać w dokumentacji dołączonej do bazy.

Przykład:

```
data DATE;
```

3.3.5 Typy danych, TIME

Pole tego typu przechowuje dane związane z godziną. Domyślnym formatem przechowywania danych jest "gg-mm-ss".

Przykład:

```
Godzina_Wypozyczenia TIME;
```

Jest jeszcze typ DATETIME będący konkatenacją daty i czasu

3.3.6 Typy danych, Text/BLOB

Powyższe typy danych używane są, jeśli zaistnieje potrzeba przechowania dłuższego ciągu znaków (z zakresu od 255 do 65535 znaków). W odróżnieniu od pól typu CHAR lub VARCHAR nie następuje tutaj obcięcie końcówki danych (chyba że ilość znaków, które chcemy umieścić w tym polu, przekracza jego górną granicę).
Jedyna różnica zachodząca między BLOB-em a TEXT-em jest taka, iż pola typu TEXT są porównywane bez rozróżnienia na wielkość liter (z ang. case insensitively), a pola typu BLOB oczywiście z uwzględnieniem wielkości liter (z ang. case sensitively).
Przykład:

```
Opis BLOB;
Opis2 TEXT;
```

3.4 Ładowanie danych do tabel

Dobrym pomysłem na załadowanie większej ilości danych jest utworzenie pliku tekstowego, w którym kolumny oddzielamy znakami tabulacji. Załóżmy, ze mamy plik pracownicy.txt, który wygląda następująco:

1 Michał Turek wykładowca
2 Antoni Ligęza profesor
3 Tomasz Szmuc profesor
4 Radosław Klimek adiunkt

Aby załadować dane z tego pliku do tabeli:

```
LOAD DATA LOCAL INFILE "pracownicy.txt" INTO TABLE Pracownicy;
```

Uwaga: jeśli utworzyliśmy ten plik pod systemem Windows, który jako znaków końca lini używa \r\n to należy posłużyć się komendą:

```
LOAD DATA LOCAL INFILE "pracownicy.txt" INTO TABLE pet LINES TERMINATED BY '\r\n';
```

Inaczej:

```
LOAD DATA LOCAL INFILE "c:/katalog/dane/pracownicy.txt" INTO TABLE Pracownicy LINES TERMINATED BY '\r\n' (Imie, Nazwisko, Stanowisko);
```

lub:

```
LOAD DATA LOCAL INFILE "c:\\katalog\\dane\\pracownicy.txt" INTO TABLE Pracownicy LINES TERMINATED BY '\r\n' (Imie, Nazwisko, Stanowisko);
```

4 Przykłady

4.1 Przykładowa baza danych sklep

Utworzenie przykładowej tabeli

```
CREATE TABLE sklep (
 artykul INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
 dostawca  CHAR(20) DEFAULT '' NOT NULL,
 cena   DOUBLE(16,2) DEFAULT '0.00' NOT NULL,
 PRIMARY KEY(artykul, dostawca));
INSERT INTO sklep VALUES
     (1,'A',3.45),
     (1,'B',3.99),
     (2,'A',10.99),
     (3,'B',1.45),
     (3,'C',1.69),
     (3,'D',1.25),
     (4,'D',19.95);
```

Po wykonaniu takich poleceń SQL mamy następującą tabelę:

Artykuł	Dostawca	Cena
0001	A	3.45
0001	B	3.99
0002	A	10.99

0003	B	1.45
0003	C	1.69
0003	D	1.25
0004	D	19.95

4.1.1 Maksymalna wartość w kolumnie

```
SELECT MAX(artykul) AS artykul_max FROM sklep;
```

4.1.2 Wiersz z największą wartością kolumny

Przy użyciu podzapytania:

```
SELECT artykul, dostawca, cena
FROM   sklep
WHERE  cena=(SELECT MAX(cena) FROM sklep);
```

W dwóch krokach:

1. Pobranie największej wartości w kolumnie:
   ```
   SELECT MAX(cena) FROM sklep;
   ```
   Zostanie zwrócona wartość 19.95
2. Posługując się wartością z punktu 1 tworzymy kolejne zapytanie:
   ```
   SELECT artykul, dostawca, cena
   FROM   sklep
   WHERE  cena=19.95;
   ```

Inne rozwiązanie może polegać na posortowaniu względem żądanej kolumny i pobranie tylko jednego ineresującego nas wiersza (w mysql wykorzystjue się do tego celu klauzulę limit):

```
SELECT artykul, dostawca, cena
FROM   sklep
ORDER BY cena DESC
LIMIT 1;
```

4.1.3 Maksymalna wartość w grupie

Który artykuł z każdej grupy ma najwyższą cenę:

```
SELECT artykul, MAX(cena) AS cena
FROM   sklep
GROUP BY artykul
```

Wynik:

artykul	Cena
0001	3.99
0002	10.99
0003	1.69
0004	19.95

4.1.4 Wiersze z maksymalną wartością w grupie

Dla każdego artykułu znajdź dostawców z najwyższą ceną.

W jednym zapytaniu:

```
SELECT artykul, dostawca, cena
FROM   shop s1
WHERE  cena=(SELECT MAX(s2.cena)
             FROM shop s2
             WHERE s1.artykul = s2.artykul);
```

W paru krokach:

1. Pobierz listę par (artykuł, cena maksymalna).

2. Dla każdego artykułu pobierz odpowiednie wiersze, które przechowują cenę maksymalną.

Rozwiązanie to można zaimplementować z użyciem tabeli tymczasowej oraz złączenia tabel:

```
CREATE TEMPORARY TABLE tmp (
 artykul INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
 cena   DOUBLE(16,2) DEFAULT '0.00' NOT NULL);

INSERT INTO tmp SELECT artykul, MAX(cena) FROM shop GROUP BY
artykul;
%Zagadka: jak to można zrobić inaczej?
SELECT shop.artykul, dostawca, shop.cena
FROM shop, tmp
WHERE shop.artykul=tmp.artykul AND shop.cena=tmp.cena;

DROP TABLE tmp;
```

Wynik:

artykul	Cena
0001	3.99
0002	10.99
0003	1.69
0004	19.95

4.2 Baza danych – zawodnicy

Przechowujemy informacje o zawdonikach z Pucharu Swiata FIS i punktach jakie zdobywają.

Utworzenie bazy danych:

```
CREATE TABLE FIS_Zawodnicy (
    ID INT NOT NULL AUTO_INCREMENT,
    Imie CHAR(20) NOT NULL,
    Nazwisko CHAR(30) NOT NULL,
    Punkty INT NOT NULL,
    Narodowosc CHAR(3) NOT NULL,
    PRIMARY KEY(ID)
);
```

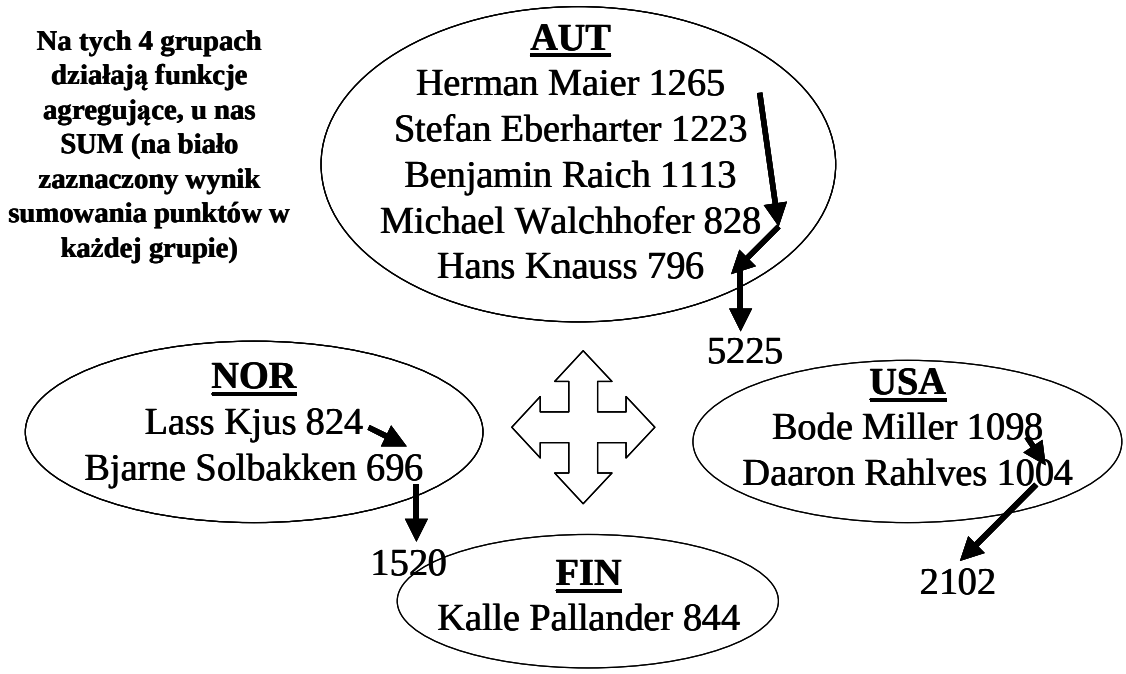
Załadowanie tabeli danymi:

```
INSERT INTO FIS_Zawodnicy (Imie, Nazwisko, Punkty, Narodowosc)
VALUES
('Herman', 'Maier', 1265, 'AUT'),
('Stefan', 'Eberharter', 1223, 'AUT'),
('Benjamin', 'Raich', 1113, 'AUT'),
('Bode', 'Miller', 1098, 'USA'),
('Daron', 'Rahlves', 1004, 'USA'),
('Kalle', 'Pallander', 844, 'FIN'),
('Michael', 'Walchhofer', 828, 'AUT'),
('Lasse', 'Kjus', 824, 'NOR'),
('Hans', 'Knauss', 796, 'AUT'),
('Bjarne', 'Solbakken', 696, 'NOR');
```

4.2.1 Klauzula GROUP BY

Załóżmy, że chcemy na podstawie danych zgromadzonych w bazie danych zrobić klasyfikację drużynową – musimy zsumować punkty zdobyte przez zawodników poszczególnych narodowości. Wynik powinien być wyświetlany w kolejności malejącej według sumy zdobytych punktów (klauzula ORDER BY). Można to zrobić przy pomocy poniższego zapytania:

```
SELECT Narodowosc, SUM(Punkty) AS „Liczba punktow”
FROM FIS_Zawodnicy
GROUP BY Narodowosc
ORDER BY “Liczba punktow” DESC
```

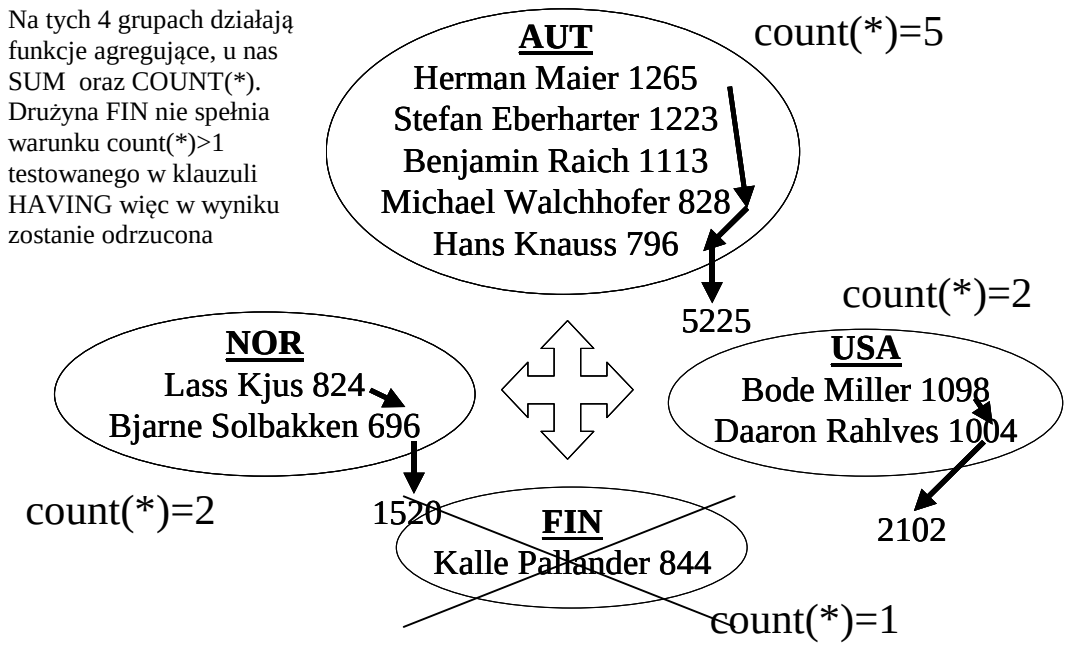


4.2.2 Klauzula HAVING

Tak jak klauzula WHERE określa predykaty służące do filtrowania wierszy, tak klauzula HAVING definiuje podobne predykaty oparte na funkcjach grupujących do odrzucania grup, które ich nie spełniają. Klauzula ta jest niezbędna do przetestowania wartości funkcji agregujących, nie pochodzących z pojedynczych wierszy iloczynu kartezjańskiego określanego przez klauzulę FROM, lecz z grup takich wierszy, a zatem nie mogących być testowanymi w klauzuli WHERE. Załóżmy zatem dla potrzeb następnego przykładu, ze do rankingu drużynowego wchodzą tylko te kraje, które mają przynajmniej dwóch zawodników.

```
SELECT Narodowosc, SUM(Punkty) AS „Liczba punktow”
FROM FIS_Zawodnicy
GROUP BY Narodowosc
HAVING COUNT(*)>1
ORDER BY “Liczba punktow” DESC
```

4.2.3 GROUP BY Narodowosc HAVING count(*)>1



4.2.4 DISTINCT

Jeżeli pobrać narodowości zawodników, którzy występują w rankingu. Wpisanie instrukcji:

```
SELECT Narodowosc FROM FIS_Zawodnicy
```

nie będzie do końca eleganckim rozwiązaniem gdyż niektóre narodowości w wyniku mogą się pojawić kilka razy.

Natomiast wpisanie:

```
SELECT DISTINCT Narodowosc
FROM FIS_Zawodnicy
```

4.2.5 Modyfikacja struktury tabeli

W celu dodania nowego pola do istniejącej tabeli lub zmiany nazwy istniejącego pola należy się posłużyć instrukcją alter table:

```
ALTER TABLE tabela add nazwa_kolumny typ_kolumny
```

Przykład:

```
ALTER TABLE FIS_Zawodnicy
ADD ID_Narty INT;
```

W celu zmiany nazwy lub typu istniejącej kolumny należy również się posłużyć instrukcją alter table:

```
ALTER TABLE table
CHANGE stara_kolumna nowa_kolumna typ_danych
```

Przykład:

```
ALTER TABLE FIS_Zawodnicy
CHANGE Punkty Liczba_Punktow
INT NOT NULL
```

4.2.6 Zbieranie informacji z więcej niż jednej tabeli

Dodajemy teraz nową tabelę do bazy danych:

```
CREATE TABLE Narty (
     ID INT AUTO_INCREMENT,
     Producent VARCHAR(40),
     Model VARCHAR(40),
     PRIMARY KEY(ID)
);
```

Zadanie: wypisz imię, nazwisko oraz producenta nart na jakich jeździ każdy zawodnik:

```
     SELECT Imie, Nazwisko, Producent  FROM FIS_Zawodnicy
     INNER JOIN Narty
     ON FIS_Zawodnicy.ID_Narty = Narty.ID
```

4.2.7 Operacje na stringach

Możemy tworzyć również wzorce według, których odbywa się wyszukiwanie – chcemy pobrać dany wszystkich zawodników, których nazwiska zaczynają się na „Ma”. Znak ‘%’ zastępuje dowolny łańcuch znaków:

```
SELECT * FROM Zawodnicy
Where Nazwisko LIKE “Ma%”
```

5 Zarządzanie użytkownikami – skrótowo

Ze względów bezpieczeństwa powinniśmy ograniczać dostęp do danych, które przechowujemy. Informacje o uprawnieniach użytkowników mysql przechowuje w bazie danych mysql. Możemy operować na danych zapisując informacje uprawnieniach bezpośrednio do odpowiednich tabel posługując się komendami SQL. Musimy jednak pamiętać aby każdorazowo potem wywoływać komendę:

```
flush privileges;
```

aby zmiany miały efekt.

Nie musimy tego robić gdy posługujemy się komendami **grant** oraz **revoke**.

Przykład:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@localhost
    ->     IDENTIFIED BY 'haslo' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@'%'
    ->     IDENTIFIED BY 'haslo' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO admin@localhost;
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

Powyższe komendy tworzą 3 użytkowników:

1. **monty** – superuser, który może się łączyć z każdego miejsca ale musi używać hasła.
2. **admin** – może się łączyć z hosta lokalnego bez hasła i ma przywileje administracyjne: reload, process

3. **dummy** – może się łączyć tylko z hosta lokalnego i nie ma danych żadnych uprawnień (opcja **USAGE**), zakłada się, ze wszelkie uprawnienia zostaną nadane później.

To samo możemy zrobić na inny sposób:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
VALUES('localhost','monty',PASSWORD('some_pass'),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
VALUES('%','monty',PASSWORD('some_pass'),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
       VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

Ustawienie hasła istniejącemu użytkownikowi:

```
SET PASSWORD FOR pawel@"%" = PASSWORD('bazy danych');
```

Uwaga: jezeli ustawiamy hasło w momencie tworzenia użytkownika komenda GRANT to użycie funkcji PASSWORD nie jest konieczne (szyfrowanie i tak jest realizowane).

W celach bezpieczeństwa dobrze jest zlikwidować użytkownika anonimowego, który może się łączyć z bazą danych bez hasła. Ja zawsze po instalacji robię:

```
mysql> use mysql;
mysql> delete from user where user = ‘’;
mysql> flush privileges;
```

6 Typy tabel w MySQL

MySQL umożliwia przechowywanie i operowanie na danych przy pomocy kilku różnych silników (storage engines):

- Oryginalnym i zaimplementowanym na początku był typ ISAM, który został następnie zastąpiony przez MyISAM, w tym trybie nie ma obsługi transakcji

Począwszy od wersji 3.23.0 wprowadzono nowe typy tabel:

- HEAP (wprowadza tabele in=memory), później przemianowany na MEMORY
- MERGE, pozwala na dostęp do zbioru identycznych tabel jak do jednej
- InnoDB oraz BDB, zapewniają obsługę transakcji, klucze obce (domyślny w wersji 5)
- NDBCluster używany do implementacji tabel, które mają być „rozproszone” na wielu komputerach

6.1 Tworzenie tabel określonego typu

Tworząc tabelę można określić jej typ dodając dyrektywę ENGINE lub TYPE:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
CREATE TABLE t (i INT) TYPE = MEMORY;
```

Uwagi:

- Począwszy od wersji 4.0.18 zaleca się używanie dyrektywy ENGINE
- Dyrektywa TYPE dostępna od wersji 3.23.0
- Jeśli nie określimy typu domyślnym jest MyISAM

6.2 Konwersja typu tabeli

Aby dokonać konwersji typu tabeli należy posłużyć się jednym z poleceń:
ALTER TABLE t ENGINE = MYISAM;
ALTER TABLE t TYPE = InnoDB;

6.3 Problemy z tabelami MyISAM

Pomimo, iż format ten jest dosyć stabilny – wszystkie zmiany w tabeli robione przez zapytanie SQL są wprowadzane przed jego zakończeniem to można doświadczyć problemów gdy:

- Proces mysqld zostanie „zabity” w trakcie zapisu
- Nieoczekiwane wyłączenie komputer’a podczas zapisu
- Błędy sprzętowe
- Używany jest zewnętrzny program (np.. myisamchk) podczas pracy serwera
- Bug softawerowy, który jeszcze nie został wykryty

Objawy błędów:

- Podczas operacji selekcji występuję błąd: Incorrect key file for table: '...'. Try to repair it
- Zapytania nie znajdują wierszy lub zwracają niepełne dane

6.4 Typ MERGE

Używany gdy chcemy używać kolekcji identycznych tabel jak jednej. Identyczność rozumiana jest jako identyczność informacji dotyczącej kolumn i indeksów. W miarę stabilny. By używać tego typu trzeba mieć uprawnienia SELECT, DELETE, UPDATE, INSERT na tych tabelach.

Przykład:

```
CREATE TABLE t1 (
     a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     message CHAR(20)
);
CREATE TABLE t2 (
     a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     message CHAR(20)
);
INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
CREATE TABLE total (
     a INT NOT NULL AUTO_INCREMENT,
     message CHAR(20), INDEX(a)
)
TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Można zrobić teraz coś takiego:

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

Uwagi:

- Kolumna a jest indeksowana w tabeli MERGE lecz nie jest zadeklarowana jako PRIMARY KEY tak jak to jest w łączonych tabelach MyISAM
- Tak musi być gdyż tabela ta nie może wymuszać unikalności tego pola na zbiorze tabel podrzędnych

MERGE, Zalety:

Można wykorzystywać gdy:

- trzymamy logi w różnych tabelach i w celu przeglądania chcemy je umieścić w jednej,
- pozwala na wzrost wydajności jeśli rozdzielimy dużą tabelę na kilka mniejszych
- zamiast złączania kilku identycznych tabel -> wzrost wydajności
- Przekroczymy maksymalny dopuszczalny rozmiar dla tabeli MyISAM (rozmiar ten podlega ograniczeniom)

MERGE, wady i problemy:

- Możliwy do użycia tylko z identycznymi tabelami MyISAM
- Zużywa więcej deskryptorów plików
- Wolniejszy odczyt pół kluczowych
- Gdy zmieniamy typ tracimy mapowanie do tabel, do których się odwołujemy a kopiowane są rekordy
- Przed wersją 4.1.1 wszystkie tabele musiały być w tej samej bazie danych
- Nie działa instrukcja REPLACE
- Używanie instrukcji RENAME TABLE na aktywnej tabeli MERGE może spowodować jej uszkodzenie
- Podczas tworzenia tabeli nie jest sprawdzane czy podrzędne tabele istnieją
- Kolejność indeksów we wszystkich tabelach musi być ta sama
- W systemie Windows nie działa instrukcja DROP TABLE na tabeli podrzędnej

6.5 Typ HEAP

Pozwala na tworzenie tabel, których zawartość jest przechowywana w pamięci. Od wersji 4.1 nazwa MEMORY jest nazwa preferowaną. Doskonale nadają się jako tabele tymczasowe. Nie wspierają pół BLOB oraz TEXT. Przed wersją 4.1.0 nie wspierały pól AUTO_INCREMENT natomiast przed wersją 4.0.2 nie wspierały indeksów na polach, które dopuszczały wartość NULL. Aby zwolnić pamięć używaną prze tabele należy wykonać instrukcję DELETE, TRUNCATE TABLE lub DROP TABLE.

6.6 Typ InnoDB

6.6.1 Transakcje, przypomnienie

Jest to grupa operacji na danych, która musi zostać wykonana cała lub w ogóle. Nie może się zdarzyć tak by tylko część operacji wchodzących w skład transakcji została wykonana. Często systemy umożliwiają jednoczesne przetwarzanie kilku transakcji (np. podejmowanie gotówki z wielu bankomatów). Poprawność przeprowadzenia transakcji opisują poniższe właściwości:

- Niepodzielność (atomicity). Żądamy albo cała transakcja została przeprowadzona albo żeby wcale nie została przeprowadzona tj. żaden z jej elementów nie zostanie uwzględniony. Przykład: podjęcie pieniędzy z bankomatu i powiązany z tym zapis tego na koncie klienta.
- Spójność (consistency). Dane muszą zaspokajać nasze oczekiwania. Przykład: system obsługi linii lotniczych: jedno miejsce w konkretnym rejsie nie zostanie przydzielone dwóm osobom.
- Izolacja (isolation). Gdy 2 transakcje są przetwarzane jednocześnie ich wyniki nie mogą na siebie wzajemnie wpływać. W wyniku tej

równoległości nie może zajść nic co by nie zaszło jakby były przetwarzane sekwencyjnie.

- Trwałość (durability). Jeśli transakcja się zakończy to jej wynik nie może zostać utracony nawet jeśli awaria nastąpi tuż po zakończeniu tej transakcji.

6.6.2 Cechy typu InnoDB

Zapewnia obsługę transakcji (zgodność z ACID) – obsługa: ROLLBACK, COMMIT, oraz crash recovery. Dokonuje blokady na poziomie wiersza oraz możliwość nie blokującego odczytu instrukcją SELECT (wzorowane na Oracle'u) – zwiększą to wydajność oraz wielodostęp. Zapewnia obsługę kluczy obcych: FOREIGN KEY. Dostępny defaultowo od wersji 4.0 MySQL'a. Od wersji 5.0 instaler na windows czyni go domyślnym typem tabel.

Domyślnie każdy serwer MySQL uruchamiany jest z opcją AUTO COMMIT, która oznacza, że po każdej instrukcji robiony jest COMMIT. Aby umożliwić wykonywanie transakcji składających się z kilku instrukcji należy wyłączyć ten tryb instrukcją: SET AUTOCOMMIT = 0 a następnie posługiwać się instrukcją COMMIT i ew. ROLLBACK.

Przed wersją 4.0.11 w celu rozpoczęcia transakcji należało używać instrukcji BEGIN zamiast START TRANSACTION.

6.6.3 Przykład

Dwie transakcje, jedna została wykonana a druga cofnięta:

```
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
TYPE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (10, ‘Test');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+------+--------+
| A    | B      |
+------+--------+
|   10 | Test   |
+------+--------+
1 row in set (0.00 sec)
```

6.6.4 Klucze obce

Obsługiwane przez typ tabel InnoDB od wersji 3.23.44. Dla innych tabel klauzula FOREIGN KEY jest poprawnie parsowana lecz jest ignorowana. Planowane jest dołączenie tej informacji w pliku specyfikującym tabele by można ją było odtworzyć. W dalszej przyszłości mają być obsługiwane również przez typ MyISAM.

Zalety używania:

- Przy założeniu poprawnego projektu bazy danych utrudniają programiści/użytkownikowi zakłócanie spójności danych
- Ponieważ sprawdzanie poprawności kluczy odbywa się po stronie serwera nie trzeba tego robić w aplikacji
- Posługiwanie się kaskadową aktualizacją lub usuwaniem upraszcza kod aplikacji
- Poprawnie zaprojektowane pełnią rolę dokumentacyjną dla związków pomiędzy tabelami.

Uwagi:

- Ponieważ sprawdzanie poprawności odbywa się po stronie serwera ma to wpływ na wydajność. Niektóre komercyjne aplikacje z tego powodu celowo robią to po stronie aplikacji (odciążenie serwera)
- Jeśli zdecydujemy się poprawność kluczy obcych realizować samemu musimy mieć na uwadze kolejność wstawiania wierszy i unikać wstawiania „osieroconych” wierszy potomnych
- Jeżeli posługujemy się tylko więzem ON DELETE to należy pamiętać, że jest możliwość usuwania jedną instrukcją DELETE danych z kilku tabel (zatem zyskujemy na wydajności).

Problemy:

- Projektując należy pamiętać o poprawności związków: unikać zapętleń oraz niewłaściwych kombinacji kaskadowych deletów
- Często się natrafia na problemy administracyjne: ciężko np. odzyskać z backupu pojedyncze tabele

6.6.4.1 Składnia

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...) [ON DELETE {RESTRICT
| CASCADE | SET NULL | NO ACTION | SET DEFAULT}] [ON UPDATE
{RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
```

Uwagi:

- Obie tabele muszą być typu InnoDB
- Korespondujące pola musza mieć ten sam typ danych tak by mogły być porównywane bez żadnej konwersji typów
- Wielkość i znak (lub brak) dla typu INTEGER musi być również taka sama!
- Długości stringów nie musza być takie same

- Jeśli się zadeklaruje pole będące referencją jako NOT NULL nie wykonywać akcji SET NULL
- Niezgodność ze standardem SQL: jeśli w macierzystej tabeli jest kilka wierszy z tę samą referencjonowaną wartością wtedy InnoDB przy sprawdzeniu zachowuję się tak jakby tych innych nie było. Jeśli użyto RESTRICT i jest taki wiersz potomny, który posiada więcej niż jednego rodzica to InnoDB nie pozwala na usunięcie żadnego z nich.

Przykład 1:

```
CREATE TABLE parent(
     ID INT NOT NULL,
     PRIMARY KEY (id)
) TYPE=INNODB;
CREATE TABLE child(
     id INT,
     parent_id INT,
     INDEX par_ind (parent_id),
     FOREIGN KEY (parent_id) REFERENCES parent(id) ON
DELETE CASCADE
) TYPE=INNODB;
```

Przykład 2:

```
CREATE TABLE PRODUKT (
     Kategoria INT NOT NULL,
     ID INT NOT NULL,
     Cena DECIMAL,
     PRIMARY KEY(Kategoria, ID)
) TYPE=INNODB;
CREATE TABLE KLIENT (
     ID INT NOT NULL,
     PRIMARY KEY (ID)
) TYPE=INNODB;
CREATE TABLE ZAMOWIENIE (
     NR INT NOT NULL AUTO_INCREMENT,
     Produkt_Kategoria INT NOT NULL,
     Produkt_ID INT NOT NULL,
     Klient_ID INT NOT NULL,
     PRIMARY KEY(NR),
     INDEX (Produkt_Kategoria, Produkt_ID),
     FOREIGN KEY (Produkt_Kategoria, Produkt_ID)
          REFERENCES PRODUKT(Kategoria, ID)
     ON UPDATE CASCADE ON DELETE RESTRICT,
     INDEX (Klient_ID),
     FOREIGN KEY (Klient_ID)
          REFERENCES KLIENT(ID)
) TYPE=INNODB;
```

Uwagi:

- Należy pamiętać o tym by najpierw dodawać potrzebne indeksy
- Można dodać referencję używając instrukcji ALTER TABLE:

```
ALTER TABLE table_name ADD [CONSTRAINT symbol] FOREIGN KEY
[id] (index_col_name, ...) REFERENCES table_name2
(index_col_name, ...) [ON DELETE {RESTRICT | CASCADE | SET
NULL | NO ACTION | SET DEFAULT}] [ON UPDATE {RESTRICT |
CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
```

- Można również usunąć referencję (od 4.0.13):
- ALTER TABLE table_name
- DROP FOREIGN KEY fk_symbol;
- Można też dodać referencję do samej siebie korzystając z alter table
- Jeśli FOREIGN KEY zawierała sekcję CONSTRAINT z nazwą to usuwając można się nią posłużyć. Jeśli nie podamy nazwy to jest generowana automatycznie wartość fk_symbol – by ją odzyskać należy posłużyć się poleceniem SHOW CREATE TABLE (tak jak w przykładzie 3)
- We wcześniejszych wersjach (3.23.x) nie należy używać poleceń ALTER TABLE lub CREATE INDEX dla tabel, które posiadają referencję lub są referencjonowane

Przykład 3:

```
mysql> SHOW CREATE TABLE test_c
******************* 1. row *******************
       Table: test_c
Create Table: CREATE TABLE `test_c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `test_a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `test_a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY 0_38775;
```

7 Stored procedures i funkcje

Dostępne począwszy od wersji 5.0. Podobne do programów w konwencjonalnych językach programowania, zbiór instrukcji SQL, który jest przechowywany w serwerze. Kiedy są pożyteczne:

- Kiedy istnieje kilka aplikacji klienckich napisanych w różnych językach, które robią to samo
- Ze względów bezpieczeństwa. Banki używają procedur/funkcji przechowywanych dla wszystkich popularnych operacji. Zapewnia to poprawne logowanie każdej operacji o dostarcza stabilnego i bezpiecznego środowiska. Aplikacje i użytkownicy mają dostęp do bazy TYLKO poprzez te procedury
- Ograniczają ilość informacji przesyłaną pomiędzy klientem a serwerem

Składnia MySQL wzorowana jest na IBM DB2. Rozróżniamy dwa rodzaje:

- procedury, mogą posiadać parametry wejściowe i wyjściowe
- funkcje, zwracają wartość, można ich używać wszędzie tam gdzie używamy predefiniowanych funkcji mysql

Warto używać tych mechanizmów gdyż:

- Są w standardzie SQL'a od dawna i są wspierane przez duże komercyjne systemy BD takie jak Oracle, DB2 (składnia podobna jeśli nie identyczna)
- Architektura komponentowa – możemy zmienić język aplikacji i nie stanowi to problemu – logika jest w bazie danych
- Przenośność, jeżeli piszemy procedurę to mamy gwarancję, że będzie działać na każdej platformie z MySQL'em. Nie potrzebujemy ustawiać uprawnień w systemie op., instalować dodatkowe oprogramowania itp.
- Sensowne i eleganckie jest łączenie w BD danych, które są przechowywane z operacjami, które na tych danych są wykonywane

Instalacja ze starszej wersji serwera:

- Jeśli nadinstalowaliśmy nowe wersję serwera nad starą (np. 4.1.10) to zostaje stary katalog data, by poprawnie obsługiwać st. proc. i fkcje to potrzebujemy w bazie mysql specjalnej tabeli proc
- W tym celu należy jako root uruchomić skrypt:

mysql>source usr/mysql50/scripts/mysql_prepare_privilege_tables_for_5.sql

- Inna opcja to wykorzystanie skryptów: mysql_fix_privilege_tables lub mysql_install_db (z podkatalogu scripts mysql'a)

7.1 Pierwszy przykład

```
CREATE PROCEDURE procedure1 /* nazwa */
(IN parameter1 INTEGER) /* parametry */
BEGIN /* początek bloku */
     DECLARE var1 CHAR(10); /* deklaracja zmiennej lokalnej */
     IF parameter1 = 17 THEN /* początek IF'a */
          SET var1 = 'rakiety'; /* przypisanie */
     ELSE SET var1 = 'piłki'; /* przypisanie */
     END IF; /* koniec IF'a */
     INSERT INTO Sprzet VALUES (var1);/* instrukcja SQL */
END /* koniec bloku */
```

Pełny prosty przykład:

```
mysql> CREATE DATABASE pierwsza;
Query OK, 1 row affected (0.01 sec)
mysql> USE pierwsza;
Database changed
mysql> CREATE TABLE test (ID INT, Imie CHAR(15));
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO test VALUES (1, 'Pawel');
Query OK, 1 row affected (0.00 sec)
mysql> delimiter //
mysql> create procedure myprocedure1() select * from test;//
Query OK, 0 rows affected (0.02 sec)
mysql> call myprocedure1()//
+------+-------+
| ID   | Imie  |
+------+-------+
|    1 | Pawel |
+------+-------+
```

7.2 Składnia

Do utworzenia należy posłużyć się instrukcjami: CREATE PROCEDURE lub CREATE FUNCTION. Wywołuje się instrukcją CALL. W wersji 5.0.0 są globalne i nie związane z bazą danych a od w wersji 5.0.1 już związane z bazą danych. Kiedy wywoływana jest procedura/funkcja implicite wywoływane jest wcześnie USE nazwa_bazy (USE wewnątrz ciała nie jest dopuszczalne). Można poprzedzać nawę procedury nazwa bazy np.: CALL test.p() (wywołanie procedury p zdefiniowanej dla bazy test). Kiedy baza jest usuwana (wersja 5.0.1) usuwane są wszystkie procedury i funkcje z nią związane.

CREATE PROCEDURE sp_name ([parametr[,...]])
  [charakterystyka ...] ciało
CREATE FUNCTION sp_name ([parameter[,...]])
  [RETURNS typ]
  [charakterystyka ...] ciało
parametr:
  [IN | OUT | INOUT] param_name type

typ:
   dowolny typ MySQL
charakterystyka:
   LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | SQL SECURITY {DEFINER | INVOKER}
  | COMMENT 'string'
ciało:
   instrukcje SQL

Uwagi:

- Domyślnie procedura/funkcja jest kojarzona z aktualną bazą danych
- Aby skojarzyć z inną należy poprzedzić nazwę nazwą bazy danych np..: test.p
- Klauzula RETURNS może być użyta tylko dla funkcji
- Nie ma jeszcze przywileju GRANT EXECUTE (tzn. nie można pozwolić jednym użytkownikom na wykonywanie procedur a innym nie)
- Klauzula COMMENT jest rozszerzeniem MySQL'a (nie ma w standardzie SQL) i służy do opisu tego co procedura robi (takie jest zamierzenie)

7.3 Co jest dozwolone a co nie

Nie możemy wewnątrz procedury używać instrukcji use. Możemy używać: select, insert, update, delete itd. – należy pamiętać, że używanie jakichkolwiek rozszerzeń mysql powoduje, że kod przestaje być przenośny. Nie możemy zagnieżdżać definicji procedur a więc nie możemy wewnątrz używać: create procedure, drop procedure, create/drop function. Nie można też używać: create/drop trigger.

7.4 Przykład procedury z parametrami

W przykładzie dokonano zmiany separatora komend z ';' na '//' aby móc się posłużyć znakiem ';' przy definicji ciała procedury zmienne poprzedzone znakiem @ to zmienne sesyjne.

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @a;
+------+
```

```
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

7.5 Charakterystyki

CREATE PROCEDURE p2 () LANGUAGE SQL //1
NOT DETERMINISTIC //2
SQL SECURITY DEFINER //3
COMMENT ‘Procedura‘ //4
SELECT * FROM test;

1. Oznacza, że językiem używanym wewnątrz procedury jest SQL (co jest bezsensu bo zawsze tak jest)
2. Klauzula informacyjna. Procedura deterministyczna dla tego samego wejścia zwraca zawsze to samo wyjście. U nas jest select * wiec wyjście zależy od zawartości tabeli test
3. Mówi, że „przy wywołaniu (call) sprawdź przywileje użytkownika, który utworzył procedurę” (inna opcja to invoker – sprawdzanie przywilejów wywołującego).
4. Komentarz jest opcjonalny – dokumentuje to co robi procedura (niezgodność ze standardem SQL2003)

Uwagi: przedstawione wyżej charakterystyki są domyślne

7.6 Parametry wejściowe procedury

```
mysql> CREATE PROCEDURE p5(p INT) SET @x = p //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL p5(12345)//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x//
+-------+
|@x     |
+-------+
| 12345 |
+-------+
1 row in set (0.00 sec)
```

Przykład 2:

```
CREATE TABLE Zamowienie (
  id int auto_increment,
  nazwa char(20),
  cena decimal(5,2),
  liczba_sztuk int,
  naleznosc decimal(5, 2),
  primary key(id)
);
create procedure
InsertZamowienie(n char(20), p decimal(5,2), q int)
     insert into Zamowienie(nazwa, cena, liczba_sztuk,
naleznosc)
     values (n, p, q, p*q);
```

7.7 Parametry wyjściowe

```
mysql> CREATE PROCEDURE p6 (OUT p INT)
    -> SET p = -5;
mysql> CALL p6(@y);
mysql> SELECT @y;
+------+
| @y   |
+------+
| -5   |
+------+
```

7.8 ALTER PROCEDURE i ALTER FUNCTION

Używamy by zmienić nazwę lub charakterystykę funkcji lub procedury, składnia:

ALTER {PROCEDURE | FUNCTION} sp_name [charakterystyka ...]

charakterystyka:

NAME nowa_nazwa

| SQL SECURITY {DEFINER | INVOKER}

| COMMENT 'string'

7.9 Przykład funkcji

Funkcja pobiera jeden parametr, przeprowadza operację wykorzystując zapytanie SQL i zwraca wynik.

```
mysql> delimiter //
mysql> CREATE FUNCTION zaproszenie (s CHAR(20)) RETURNS
CHAR(50)
    -> RETURN CONCAT('Zapraszamy na wykład, ', s, '!');
    -> //
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
mysql> SELECT zaproszenie('Pani Kasiu');
```

```
+-------------------------------------+
| zaproszenie('Pani Kasiu')           |
+-------------------------------------+
| Zapraszamy na wykład, Pani Kasiu!   |
+-------------------------------------+
1 row in set (0.00 sec)
```

7.10 Usuwanie procedur/funkcji

Składnia:

```
     DROP {PROCEDURE | FUNCTION}
     [IF EXISTS] sp_name
```

Sprawdzanie – rozszerzenie MySQL, analogicznie jak dla tablic, składnia:

```
     SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

7.11 Sprawdzanie statusu

Rozszerzenie MySQL. Zwraca informacje o procedurze/funkcji o określonym wzorcu nazwy jeśli jest zadany (jeśli nie to zwraca informacje o wszystkich procedurach/funkcjach).

Składnia:

```
     SHOW {PROCEDURE | FUNCTION}
     STATUS [LIKE 'pattern']
mysql> SHOW FUNCTION STATUS LIKE ‘zaproszenie'\G
*********** 1. row ******************
     Db: test
     Name: zaproszenie
     Type: FUNCTION
      Definer: skrzynia@localhost
     Modified: 0000-00-00 00:00:00
     Created: 2004-12-14 22:29:37
     Security_type: DEFINER
     Comment:
```

7.12 Instrukcja CALL

Składnia:

```
     CALL sp_name([parameter[,...]])
```

Służy do wywołania procedury, która wcześniej została zdefiniowana przy pomocy CREATE PROCEDURE. Może zwracać wartości poprzez parametry.

7.13 Instrukcja grupująca BEGIN ... END

Składnia:

```
     [begin_label:] BEGIN
     statement [statement] ...
     END [end_label]
```

Dzięki temu ciało procedury/funkcji może zawierać więcej niż jedną instrukcję SQL.

7.14 Instrukcja DECLARE

Używana do deklaracji składników lokalnych w ciele procedury/funkcji: zmienne lokalne, warunki i obsługa, kursory. Może być używane tylko wewnątrz bloku BEGIN END i to na samym początku zanim zaczną się inne instrukcje.

Definicja zmiennej loklanej, składnia:

DECLARE var_name[,...] type
[DEFAULT value]

Zasięg: wewnątrz bloku BEGIN ... END

Ustawienie zmiennej, składnia:

SET var_name = expr [, var_name = expr] ...

Rozszerza standardowy SET, pozwala jednorazowo przypisać zmienne różnych typów. Przykład:

SET a=b, x=y;

```
CREATE PROCEDURE p8 (IN we INT)
BEGIN
  DECLARE a INT; //nie ma wart. domyślnej!!!
  DECLARE b INT; //tutaj też nie ma!!!
  SET a = we;
  SET b = 5;
  INSERT INTO t1 VALUES (a);
  SELECT ID * a FROM t1 WHERE ID >= b;
END; //
```

Zasięg zmiennych:

```
CREATE PROCEDURE zasieg ()
BEGIN
  DECLARE x1 CHAR(9) DEFAULT ‘wewnatrz';
  BEGIN
    DECLARE x1 CHAR(9) DEFAULT ‘zewn.';
    SELECT x1;
  END;
  SELECT x1;
END; //
```

Wynik:

```
mysql> call zasieg()//
+-------+
| x1    |
+-------+
| zewn. |
+-------+
1 row in set (0.00 sec)
+----------+
| x1       |
+----------+
| wewnatrz |
+----------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.06 sec)
```

7.15 Kursory

Mogą być implementowane wewnątrz procedur i funkcji
Właściwości:

- asensitive, serwer może lub nie zrobić kopie rezultatu
- Read-only
- non-scrolling

Składnia:

DECLARE cursor_name CURSOR FOR sql_statement

Może być ich kilka w ciele lecz nazwy muszą być unikalne
Przykład:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  DECLARE cur1 CURSOR FOR SELECT id, data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE a CHAR(16);
  DECLARE b, c INT;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
       IF b < c THEN
          INSERT INTO test.t3 VALUES (a,b);
       ELSE
          INSERT INTO test.t3 VALUES (a,c);
       END IF;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
  CLOSE cur2;
END
```

Otwarcie zadeklarowanego kursora:

OPEN cursor_name

Pobranie kolejnego wiersza i inkrementacja wskaźnika bieżącego wiersza:

FETCH cursor_name INTO var_name [, var_name] ...

Zamknięcie kursora (robione automatycznie gdy wskaźnik dojdzie do końca):

CLOSE cursor_name

7.16 Przepływ sterowania

Wspierane są instrukcje: IF, LOOP, CASE, WHILE, ITERATE, LEAVE, REPEAT. Nie ma jeszcze wsparcia dla pętli FOR. Każda z tych instrukcji może zawierać pojedynczą instrukcje lub grupę jeśli jest ona umieszczona w instrukcji grupującej BEGIN ... END.

7.16.1 IF

Składnia:

```
IF warunek THEN instrukcj-a/e
    [ELSEIF warunek THEN instrukcj-a/e]
    ...
    [ELSE instrukcj-a/e]
END IF
```

Semantyka: jeśli warunek jest spełniony wykonywana jest instrukcj-a/e, jeśli nie to instrukcj-a/e po słowie kluczowym else

Uwaga: jest też funkcja IF() (przykład: IF(1>2,2,3) zwraca 3)

Przykład:

```
CREATE PROCEDURE p_if (IN par1 INT)
BEGIN
  DECLARE var1 INT;
  SET var1 = par1 + 1;
  IF var1 = 0 THEN
    INSERT INTO t1 VALUES (17);
  END IF;
  IF par1 = 0 THEN
    UPDATE t1 SET ID = ID + 1;
  ELSE
    UPDATE t1 SET ID = ID + 2;
  END IF;
END; //
```

7.16.2 CASE

Składnia:

```
CASE case_value
    WHEN when_value THEN statement
    [WHEN when_value THEN statement ...]
    [ELSE statement]
END CASE
```

lub:

```
CASE
    WHEN search_condition THEN statement
    [WHEN search_condition THEN statement ...]
    [ELSE statement]
END CASE
```

Implementuje złożoną konstrukcję warunkową

Przykład:
```
CREATE PROCEDURE ZamowienieStandard (IN nr INT)
BEGIN
  CASE nr
    WHEN 0 THEN
      CALL InsertZamowienie(‘Tourna’, 24.99, 1);
    WHEN 1 THEN
      CALL InsertZamowienie(‘Babolat VS’, 26.99, 1);
    ELSE
      CALL InsertZamowienie(‘Tourna’, 23.99, 10);
  END CASE;
END; //
```

7.16.3 LOOP i LEAVE

LOOP, składnia:
```
[begin_label:] LOOP
    statement(s)
END LOOP [end_label]
```
Prosta pętla, powtarzana jest instrukcja aż do wyjścia (które można osiągnąć za pomocą LEAVE). Uwaga: begin_label i end_label muszą być takie same!!!

LEAVE, składnia:

LEAVE label

Używane do wyjścia z dowolnego przepływu sterowania

7.16.4 REPEAT

Składnia:
```
[begin_label:] REPEAT
     instrukcja
UNTIL warunek
END REPEAT [end_label]
```
begin_label i end_label muszą być takie same!!! Powtarzane są instrukcje tak długo jak warunek nie jest spełniony.

Przykład:
```
mysql> delimiter //
mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

7.16.5 WHILE

Składnia:

```
[begin_label:] WHILE warunek DO
     instrukcja
END WHILE [end_label]
```

Powtarzane są instrukcje tak długo aż warunek jest spełniony begin_label i end_label muszą być takie same!!!

Przykład:

```
CREATE PROCEDURE p_while ()
BEGIN
  DECLARE v INT;
  SET v = 10;
  WHILE v < 15 DO
    INSERT INTO t1 VALUES (v);
    SET v = v + 1;
  END WHILE;
END; //
```

7.16.6 ITERATE

Może się pojawiać tylko przy WHILE, REPEAT i LOOP a oznacza „zrób kolejną iteracją pętli”

Przykład:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

7.16.7 Warunki (conditions), obsługa warunków/błędów

Określone warunki wymagać mogą określonej obsługi w związku z np. zaistniałymi błędami. Asocjuje nazwę z określonym warunkiem (nazwa może być równolegle używana w DECLARE HANDLER)

Składnia:

```
DECLARE condition_name
CONDITION FOR condition_value
condition_value:
     SQLSTATE [VALUE] sqlstate_value
     | mysql_error_code
```

Określone warunki wymagać mogą określonej obsługi w związku z np. zaistniałymi błędami. Asocjuje nazwę z określonym warunkiem (nazwa może być równolegle używana w DECLARE HANDLER)

Składnia:

```
DECLARE condition_name
CONDITION FOR condition_value
```

condition_value:
SQLSTATE [VALUE] sqlstate_value
| mysql_error_code

Możemy zdefiniować obsługę zaistnienia pewnych warunków (obsługi przy pomocy instrukcji).

Składnia:

DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement

handler_type:
CONTINUE
| EXIT
| UNDO

condition_value:
SQLSTATE [VALUE] sqlstate_value
| condition_name
| SQLWARNING
| NOT FOUND
| SQLEXCEPTION
| mysql_error_code

Przykład:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter //
mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2
= 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Uwagi:

- Wartość zmiennej @x wynosi 3 zatem wykonało się wszystko do końca
- Gdyby nie było lini:
  `DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;`
  Drugi insert nie wykonałby się (ze względu na klucz) a domyślnym działaniem jest wyjście i SELECT @x zwróciło by 2.

7.16.8 Przykładowy problem: logowanie błędów

Gdy zawiedzie insert chcę mieć log błędu: kiedy i dlaczego

```
mysql> CREATE TABLE t2 (
    ->  s1 INT,
    ->  PRIMARY KEY (s1)) engine=innodb;
mysql> CREATE TABLE t3 (
    ->  s1 INT,
    ->  KEY (s1),
    ->  FOREIGN KEY (s1) REFERENCES t2 (s1)
    -> ) engine=innodb;
mysql> INSERT INTO t3 VALUES (5);
... ERROR 1216 (23000): Cannot add or update a child row: a
foreign key constraint fail
```

Próba dodania do tabeli t3 wartości, której nie ma w tabeli t2 zakończy się błędem (nr 1216 lub 1452 w zależności od wersji). Uwaga ten błąd jest błędem MySQL!

Chcemy teraz logować taki fakt w tabeli

Utworzenie tabeli z logami:

```
CREATE TABLE error_log (
     error_message CHAR(80),
     error_date DATETIME
);
```

Przykładowa procedura logująca błąd:

```
CREATE PROCEDURE p_log_errors (param1 INT)
BEGIN
     DECLARE EXIT HANDLER FOR 1216, 1452
     INSERT INTO error_log VALUES (CONCAT('Foreign Key
Reference Failure For Value = ', param1), now());
     INSERT INTO t3 VALUES (param1);
END;//
```

7.17 Uwagi

— Procedura się zaczyna od deklaracji zmiennych (kolejność istotna!!!)
— Następnie może być deklaracja warunków (conditions)
— Następnie jest deklaracja kursora
— Następnie deklaracja handlerów
— Zaburzenie tej kolejności spowoduje błąd!!!

8 Wyzwalacze (triggers)

Jest to obiekt bazy danych, który posiada nazwę i jest powiązany z tabelą. Obsługa począwszy od wersji 5.0.2. Jest aktywowany w momencie gdy wykonywana jest na tabeli jedna z następujących instrukcji: INSERT, UPDATE, DELETE. Może być aktywowany przed lub po akcji

Przykład:

Trigger jest powiązany z insertem i jest wykonywany przed operacją insert i dokonuje sumowania wartości wstawianych do kolumny:

```
CREATE TABLE Rachunek (
     Numer INT,
     Saldo DECIMAL(10,2)
);
CREATE TRIGGER ins_sum BEFORE INSERT
ON Rachunek
FOR EACH ROW SET @sum = @sum + NEW.Saldo;
```

NEW.Saldo oznacza wartość atrybutu Saldo w nowo wstawianym wierszu

Aby posługiwać się triggerem należy ustawić wartość zmiennej @sum na 0:

```
     mysql> SET @sum = 0;
     mysql> INSERT INTO account VALUES (137,14.98),
(141,1937.50), (97,-100.00);
     mysql> SELECT @sum AS 'Suma operacji';
     +---------------+
     | Suma operacji |
     +---------------+
     | 1852.48       |
     +---------------+
```

Uwagi:

- Żeby usunąć trigger należy posłużyć się poleceniem drop: DROP TRIGGER ins_sum;
- Nie można mieć dla tabeli kilku triggerów o tej samej nazwie
- W kolejnych wersjach może się to jednak zmienić i nazwa triggera będzie musiała być unikalna w całej bazie danych
- Nie można mieć na tabeli 2 triggerów dla tej samej operacji i tym samym czasie aktywacji (np.. BEFORE INSERT). To nie ma sensu gdyż można po FOR EACH ROW wstawić BEGIN oraz END i włożyć tam wiele instrukcji.
- Trigger nie może się odwoływać jawnie do nazw tabel – do dyspozycji są referencje OLD oraz NEW. OLD odnosi się do aktualnego wiersza, który ma być usunięty lub zaktualizowany.
- Nie można wywoływać w triggerze procedur instrukcją CALL
- Nie można również używać instrukcji związanych z transakcjami
- Referencja OLD jest read-only i może być stosowana w triggerach dla instrukcji UPDATE oraz DELETE
- Referencja NEW nie jest read-only i może być stosowana w triggerach dla instrukcji UPDATE oraz INSERT

- W triggerach o czasie aktywacji BEFORE możemy posługiwać się poleceniem SET – oznacza to, że możemy modyfikować wartości wstawianych atrybutów
- Referencje OLD i NEW są rozszerzeniem MySQL'a standardu SQL'a

Przykład (trigger, wiele wierszy):

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE
    -> ON account
    -> FOR EACH ROW
    -> BEGIN
    ->   IF NEW.amount < 0 THEN
    ->     SET NEW.amount = 0;
    ->   ELSEIF NEW.amount > 100 THEN
    ->    SET NEW.amount = 100;
    ->   END IF;
    -> END//
mysql> delimiter ;
```

W kolejnej wersji MySQL będzie dodany przywilej: CREATE TRIGGER żeby możliwe było coś takiego:

GRANT CREATE TRIGGER ON <table name> TO <user list>;

Standard SQL dopuszcza klauzulę CHCECK w CREATE TABLE np:

```
CREATE TABLE t25 (
      s1 INT,
      s2 CHAR(5),
      PRIMARY KEY (s1),
      CHECK (LEFT(s2,1)='A')
) ENGINE=INNODB;
```

CHECK oznacza, że insert i update są niemożliwe jeśli nie spełniają warunku (czyli pierwsza litera kolumnys2 to 'A')

Niestety w MySQL nie ma jeszcze wsparcia dla tej konstrukcji lecz można ją emulować właśnie przy pomocy trigger'a:

```
CREATE TABLE t25 (
     s1 INT, s2 CHAR(5),
     PRIMARY KEY (s1)
) ENGINE=INNODB//
CREATE TRIGGER t25_bi
BEFORE INSERT ON t25
FOR EACH ROW
IF LEFT(NEW.s2,1)<>'A' THEN
     SET NEW.s1=0;
END IF;//
CREATE TRIGGER t25_bu
BEFORE UPDATE ON t25
FOR EACH ROW
IF LEFT(NEW.s2,1)<>'A' THEN
     SET NEW.s1=0;
END IF;//
```

9 Widoki

- Pozwalają na prezentację danych w formie wygodnej dla użytkownika
- Opierają się na jednej lub kilku tabelach
- Jeżeli chodzi o operację selekcji lub projekcji to korzysta się z nich jak z tabel
- Niektóre widoki są aktualizowalne (można wykonywać operacje UPDATE, DELETE)
- Niektóre pozwalają też na INSERT
- Dostępne w MySQL od wersji 5
- Widoki są używane do:
    - utworzenia dodatkowego poziomu zabezpieczenia tabeli poprzez ograniczenie dostępu do określonych kolumn lub wierszy tabeli bazowej
    - ukrycia złożoności danych - na przykład widok może być użyty do operacji na wielu tabelach tak, by wydawało się, że operacje wykonywane są na jednej tabeli.
    - pokazywania danych z innej perspektywy - dla przykładu widok może zostać użyty do zmiany nazwy kolumny bez zmiany rzeczywistych danych zapisanych w tabeli.
    - zapewnienia poziomu integralności.

9.1 Po co nam widoki

Mogą być efektywnymi kopiami tabel bazowych (synonimami tabel jak w przykładzie poniżej):

```
CREATE VIEW v AS SELECT * FROM t;
SELECT * FROM v;
```

Mogą posiadać nazwy kolumn, wyrażenia i dowolne klauzule (group by, order by, having):

```
CREATE VIEW v AS SELECT UPPER(`table1`.`column1`),
‘Cos tam' || 'q' FROM `table1`, table2
WHERE table1.column2 < 10;
```

Używanie widoków przy insert/update/delete:

```
CREATE TABLE t (col1 INT, col2 INT);
CREATE VIEW v AS SELECT * FROM t;
INSERT INTO v VALUES (1, 2);
UPDATE v SET col1 = col1 + 10;
DELETE FROM v WHERE col1 < 5;
```

Widoki z wyrażeniami w klauzuli select:

```
CREATE VIEW v AS SELECT AVG(col1) FROM t;
CREATE VIEW v AS SELECT col1 + 17 AS `Cos` FROM t;
```

Możemy też tworzyć widoki dla widoków:

```
CREATE TABLE t (s1 INT);
CREATE VIEW v1 AS SELECT * FROM t;
CREATE VIEW v2 AS SELECT * FROM v1;
CREATE VIEW v3 AS SELECT * FROM v2;
```

9.2 Upgrade z wersji 4 do 5

Aby obsługiwać poprawnie widoki należy najpierw naprawić tabelę z przywilejami, skrypt który to robi automatycznie wygląda tak:

```
UPDATE user SET
    Create_view_priv = Create_priv,
    Show_view_priv = Create_priv
WHERE
    user <> '' AND
    "Already Had Create_view_priv" = FALSE;
```

9.3 Składnia

```
CREATE [OR REPLACE]
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view-name
[(column-list)]
AS select-statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- Klauzula algorithm jest niestandardowa
- OR REPLACE również jest niestandardowe
- Nazwa widoku do 64 znaków i musi się zaczynać od znaku litery, może być poprzedzona nazwą BD oddzieloną kropką
- [(column-list)], lista nazw kolumn widoku, jeśli pominięta to nazwy są takie jak zwracane przez select

9.4 Czego nie można

- Tworzyć widoków dla tabel tymczasowych
- Tworzyć widoków o nazwach, które już są zajęte przez inne tabele/widoki
- Nie można tworzyć widoków dla tabel, które nie istnieją (czyli nie można utworzyć widoku przed tabelą bazową)
- Nie można utworzyć widoku gdy się nie ma przywileju CREATE VIEW oraz odpowiednich na talach bazowych (o tym później)
- Widoki nie mogą w żaden sposób zależeć od zmiennych lub innych parametrów
- W zapytaniu, na którym bazuje widok nie może być podzapytania w klauzuli FROM
- Nie może być duplikatów w nazwach kolumn
- Lista kolumn ([column list]) nie zgadza się z tym co zwraca select, na którym bazuje widok:

```
CREATE VIEW v (s1) AS SELECT s1, s2 FROM t;
ERROR 1353 (HY000): View's SELECT and view's field list have
different column counts
```

Kolumny widoku są do niego dodawane w momencie tworzenia widoku, zatem:

```
mysql> CREATE TABLE t (s1 INT);
mysql> CREATE VIEW v AS SELECT * FROM t;
mysql> ALTER TABLE t ADD s2 INT;
mysql> SELECT * FROM v;
+------+
| s1   |
+------+
| 1    |
+------+
1 row in set (0.00 sec)
```

Zmiany ustawień systemowych mogą mieć wpływ na to co jest wyświetlane prze widok:

```
mysql> CREATE TABLE t (c CHAR(5));
mysql> INSERT INTO t VALUES (‘ola'),(‘tomek'),(‘kazek');
mysql> CREATE VIEW v AS SELECT group_concat(c) FROM t;
mysql> SELECT * FROM v;
+------------------+
| group_concat(c)  |
+------------------+
| ola,tomek,kazek  |
+------------------+
mysql> SET group_concat_max_len = 4;
mysql> SELECT * FROM v;
+----------------+
| group_concat() |
+----------------+
|ola,            |
+----------------+
```

9.5 Przwileje

Nowe przywileje dla GRANT/REVOKE w związku z widokami:

- SHOW VIEW
- CREATE VIEW, niestandardowy przywilej (lecz w Oracle jest taki sam)

Przykład:

1. GRANT CREATE VIEW ON db1.t1 TO skrzynia;
2. GRANT CREATE VIEW ON db1.* TO mitu;
3. GRANT CREATE VIEW ON *.* TO skrzynia;

1: Użytkownik skrzynia ma prawo do utworzenia widoku o nazwie t1 w bazie db1 pod warunkiem, że ma odpowiednie przywileje na tabelach, od których widok będzie zależeć

2: Użytkownik mitu może tworzyć dowolne widoki w bazie db1

3: Użytkownik skrzynia może tworzyć dowolne widoki we wszystkich bazach

Wniosek: do utworzenia widoku przywilej CREATE VIEW nie jest wystarczający:

- Standard SQL: potrzebne uprawnienie SELECT na każdej kolumnie, do której widok się odwołuje)
- MySQL: potrzebny jakikolwiek przywilej na każdej kolumnie, do którego widok się odwołuje

Jeżeli potrzebujemy przywileju do usuwania, wstawiania i aktualizacji wierszy w tabeli to takich samych potrzebujemy dla widoku

Zasady:

- Generalna: przywileje widoku są takie jak przywileje tabel macierzystych
- Szczególna: jeżeli mam na widoku przywilej do wykonywania jakiejś operacji to nie potrzebuję takiego samego na macierzystej tabeli. Zatem mogę mieć przywilej UPDATE na widoku v bazującym na tabeli t ale nie muszę mieć tego przywileju na tabeli t

9.6 Klauzula ALGORITHM

```
CREATE [OR REPLACE]
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]
VIEW view-name ...
```

Możliwe są 2 algorytmy przy przetwarzaniu widoku:

- Łączący (MERGE), użycie takiego zapytania na widoku: SELECT * FROM v spowoduje wykonanie ekwiwalentu na tabeli macierzystej
- Bazujący na tabeli tymczasowej (TEMPTABLE), oznacza, że użycie takiego samego zapytania na widoku spowoduje utworzenie ekwiwalentnej tabeli tymczasowej, na której jest wykonane

Trzecia opcja (domyślna) pozwala MySQL na wybór najodpowiedniejszego z dwóch UNDEFINED: MERGE jest wywoływany tylko wtedy gdy jest możliwe mapowanie 1 do 1 między tabelą a widokiem (nie ma distinct, fkcji agregujących itp.) i jest trochę szybszy.

Zagadnienia te należy brać pod uwagę przy projektowaniu transakcji gdy możliwe jest blokowanie tabel.

9.7 Aktualizacja widoków

Tylko PostgreSQL (09-2005) i SQLite tego nie wspierają lecz jest to wymóg standardu. Co sprawia, że widok nie jest aktualizowalny:

1. UNION ALL
2. UNION
3. DISTINCT or DISTINCTROW
4. Podzapytanie w liście kolumn SELECT'a
5. Nieaktualizowalny widok w klauzuli FROM
6. SELECT ... FROM t WHERE x = (SELECT ... FROM t)
7. ALGORITHM = TEMPTABLE

MySQL nie spełnia wymagania CODD'a by wszystkie teoretycznie aktualizowalne widoki były praktycznie aktualizowalne (lecz jest bliski spełnienia go).
Uwagi:
ad. 1 i 2. UNION, UNION ALL

```
CREATE VIEW v AS
SELECT * FROM t1
UNION ALL
SELECT * FROM t2
```

Teoretycznie możliwe lecz w MySQL niemożliwe implementacyjnie (ze względu na użycie tabel tymczasowych do przetwarzania)

```
CREATE VIEW v AS
SELECT * FROM t1 UNION /* lub EXCEPT lub INTERSECT */
SELECT * FROM t2;
```

Takie widoki nie są aktualizowalne teoretycznie (zgodnie ze standardem SQL)
ad. 4.

```
CREATE VIEW v AS
SELECT t1.*,(SELECT s1 FROM v2)
FROM t1;
```

To ograniczenie ma zostać usunięte wkrótce
MySQL pozwala na usuwanie rekordów z widoku nawet gdy widok nie zawiera wszystkich kolumn z tabeli (może nie jest to dobre ale w innych SZBD jest podobnie)

9.7.1 Wstawianie rekordów do widoku

Widok musi zawierać każdą kolumnę tabel macierzystych, która nie ma przypisanej wartości domyślnej
Widok nie może zawierać kolumn pochodnych (np.. Będąych rezultatem obliczeń itp)
Widok nie może zawierać duplikatów np..:

```
SELECT a AS a1, a AS a2 FROM t;
```

```
CREATE TABLE t (s1 INT, s2 INT NOT NULL);
CREATE VIEW v AS SELECT s1, s2 FROM t;
UPDATE v SET s1 = 5; /* O.K. */
INSERT INTO v (s1) VALUES (1); /* zawiedzie? */
```

Deafultowo MySQL przypisuje 0 wszystkim numerycznym kolumnom z atrybutem NOT NULL więc pójdzie.
Jeśli jest ustawiony sql_mode='traditional' to będzie błąd

10 Programy narzędziowe

Ułatwiają pracę z MySQL i administację:

- mysqldump
- mysqlhotcopy
- mysqlimport
- perror
- mysqlcheck
- mysql administrator (GUI)
- mysql query browser (GUI)

10.1 mysqldump

Używany do „zrzucenia” jednej lub więcej baz danych do pliku tekstowego w celu przeniesienia go na inny serwer SQL (niekoniecznie MySQL). Może służyć do robienia backupów. Zawiera instrukcje SQL tworzące tabele oraz wypełnia je danymi. Są trzy sposoby wywołania:

shell> mysqldump [options] db_name [tables]
shell> mysqldump [options] --databases DB1 [DB2 DB3...]
shell> mysqldump [options] --all-databases

Jeśli nie nazwiemy tabel jeśli użyjemy --databases lub --all-databases zostaną zrzucone wszystkie bazy danych znajdujące się na serwerze

Jeżeli nie użyjemy opcji --quick lub --opt cała zawartość zostanie najpierw załadowana do pamięci przed zapisem do pliku (co może być problemem), począwszy od wersji 4.1 –opt jest domyślne lecz może być pominięte opcją:

--skip-opt

Przykład (zrobienie backupu całej bazy danych):

mysqldump --opt db_name > backup-file.sql

Wczytanie bazy z powrotem:

shell> mysql db_name < backup-file.sql
lub:
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name

Sklonowanie bazy danych na inny serwer:

shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name

10.2 mysqlhotcopy

Skrypt oryginalnie napisany w Perlu służący do szybkiego backupowania baz danych znajdujących się na serwerze. Można go używać tylko na maszynie, na której znajdują się katalogi z bazami danych (fizycznie).

Działa tylko dla tabel MyISAM i ISAM

Wywołanie:

shell> mysqlhotcopy db_name [/path/to/new_directory]
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_dir
shell> mysqlhotcopy db_name./regex/

W celu poznania wszystkich opcji wywołać z opcją:

--help

10.3 mysqlimport

Służy do importowania danych, skrypt oparty jest o polecenia:

LOAD DATA INFILE

Wywołanie:

mysqlimport [options] db_name textfile1 [textfile2 ...]

Dla każdego pliku program obcina jego rozszerzenie i na podstawie tego co zostaje importuje zawartość p do odpowiedniej tabeli

Przydatne opcje:

--columns=lista_kolumn, -c lista_kolumn, ta opcja pobiera listę kolumn oddzielonych przecinkami, kolejność ważna

--delete, -d, kasuje zawartośc tabeli przed importem

--help, wyświetla help

--fields-terminated-by=... , --fields-enclosed-by=... , --fields-optionally-enclosed-by=... , --fields-escaped-by=... , --lines-terminated-by=... , takie samo znaczenie jak w instrukcji LOAD DATA INFILE

10.4 mysqlshow

Służy do szybkiego sprawdzania jakie bazy danych istnieją na serwerze, jakie mają tabele, jakie pola mają tabele itp..

Wywołanie:

shell> mysqlshow [options] [db_name [tbl_name [col_name]]]

Jeśli nie podamy bazy danych wyświetlone zostaną wszystkie bazy. Jeśli nie podamy tabeli bazy danych wyświetlone zostaną wszystkie. Jeśli nie podamy kolumny tabeli wyświetlone zostaną wszystkie wraz z informacjami.

10.5 perror

Służy do wyjaśniania kodów błędów

Przykładowe wywołanie:

shell> perror 13 64

Zwróci:

Error code 13: Permission denied

Error code 64: Machine is not on the network

Składnia:

shell> perror [options] errorcode ...