Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki
Instrukcja do zajęć laboratoryjnych

Praca z bazą danych MySQL
wersja 2.0

Nr ćwiczenia:	9
Temat:	**Ograniczenia (ang. *constraints*) bazodanowe**
Cel ćwiczenia:	Celem ćwiczenia jest zapoznanie studenta z tzw. ograniczeniami bazodanowymi. Pozwalają wprowadzać na kolumny pewne dodatkowe warunki (np.: „kolumna nazwisko nie może być pusta" lub też „w polu pleć dozwolonymi znakami są 'K' oraz 'M').
Wymagane przygotowanie teoretyczne:	Wiadomości podawane na wykładzie.
Sposób zaliczenia:	Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.
Prowadzący zajęcia:	**dr inż. Artur Gramacki, dr inż. Jarosław Gramacki**

1. Uwagi wstępne

Bardzo często na kolumny w tabeli nakładamy pewne dodatkowe warunki. Przykładowo w kolumnie numerycznej możemy nakazać, aby możliwe było wpisywanie tylko liczb ze zbioru {1, 2, 3, 4}. Podobnie dla np. kolumny tekstowej możemy nakazać wpisywanie tylko i wyłącznie napisów *karta płatnicza*, *gotówka* oraz *przelew*. O kolumnach takich mówimy, że mają one zdefiniowane pewne dodatkowe warunki (nazywane właśnie *ograniczeniami*), które pozwolą nam osiągnąć tanim kosztem zamierzony efekt.

Nakładanie ograniczeń na kolumny pozwala nam przeprowadzać kontrolę wprowadzanych danych na najniższym z możliwych poziomów — na poziomie bazy danych. Oczywiście kontrolę taką można też przeprowadzić na poziomie aplikacji, jednak powinno się traktować jako dobrą zasadę programistyczną aby, jeżeli jest to tylko możliwe, przeprowadzać kontrolę wprowadzanych danych możliwie „jak najbliżej" serwera bazy danych. Statystycznie rzecz biorąc jest bowiem bardziej prawdopodobne, że to my popełnimy błąd w naszej aplikacji, niż że błąd ten powstanie w wyniku niewłaściwe działającego mechanizmu obsługi ograniczeń w serwerze bazy. Poza tym nałożenie odpowiednich ograniczeń bazodanowych jest prawie zawsze dużo mniej czasochłonne i o wiele łatwiejsze niż implementacja podobnej funkcjonalności na poziomie aplikacji.

2. Przykłady ograniczeń

2.1. Wszystkie oprócz FOREIGN KEY

Poniżej podajemy definicję przykładowej tabeli, w której zdefiniowano na podanych kolumnach różne ograniczenia:

Nazwa kolumny	Ograniczenie	Uwagi
prac_id	PRIMARY KEY + AUTO_INCREMENT	—
imie	NOT NULL + UNIQUE	UNIQUE na wszystkich trzech kolumnach (imie, nazwisko, pseudo)
nazwisko	NOT NULL + UNIQUE	
pseudo	NOT NULL + UNIQUE	
pesel	NOT NULL + UNIQUE	—
plec	NOT NULL + ENUM	dopuszczalne wartości to: *TAK* albo *NIE*
platnosci	NOT NULL + SET	dopuszczalne wartości to elementy zbioru: {*gotówka*, *karta kredytowa*, *przelew*}
czy_pracuje	DEFAULT	domyślna wartość to: *TAK*
zarobki	NOT NULL	—
data_zatr	NOT NULL	—

Definicja tej tabeli w języku SQL (w „dialekcie" MySQL-a) wygląda następująco:

```
CREATE TABLE pracownicy
(
   prac_id      INT                     PRIMARY KEY AUTO_INCREMENT,
   imie         VARCHAR(20)             NOT NULL,
   nazwisko     VARCHAR(30)             NOT NULL,
   pseudo       VARCHAR(15)             NOT NULL,
   pesel        CHAR(11)                NOT NULL UNIQUE,
   plec         ENUM ('K', 'M')         NOT NULL,
   platnosci    SET ('gotowka',
                     'karta kredytowa',
                     'przelew')          NOT NULL,
   czy_pracuje  CHAR(3)                 DEFAULT 'TAK',
   zarobki      NUMERIC(11,2)           NOT NULL,
   data_zatr    DATE,
   UNIQUE(imie, nazwisko, pseudo)
);
```

Poleceniem `DESCRIBE` oglądamy definicję tabeli i sprawdzamy, czy uzyskaliśmy to, czego oczekiwaliśmy:

```
mysql> DESCRIBE pracownicy;
+------------- +-------------------------+------+-----+---------+----------------+
| Field        | Type                    | Null | Key | Default | Extra          |
+--------------+-------------------------+------+-----+---------+----------------+
| prac_id      | int(11)                 | NO   | PRI | NULL    | auto_increment |
| imie         | varchar(20)             | NO   | MUL |         |                |
| nazwisko     | varchar(30)             | NO   |     |         |                |
| pseudo       | varchar(15)             | NO   |     |         |                |
| pesel        | char(11)                | NO   | UNI |         |                |
| plec         | enum('K','M')           | NO   |     |         |                |
| platnosci    | set('gotowka','karta... | NO   |     |         |                |
| czy_pracuje  | char(3)                 | YES  |     | TAK     |                |
| zarobki      | decimal(11,2)           | NO   |     |         |                |
| data_zatr    | date                    | YES  |     | NULL    |                |
+--------------+-------------------------+------+-----+---------+----------------+
10 rows in set (0.01 sec)
```

W MySQL-u istnieje komenda, pokazujące polecenie SQL, które tworzy daną tabelę:

```
mysql> SHOW CREATE TABLE pracownicy;
```

```
+------------+-------------------------------------------------------------
| Table      | Create Table
+------------+-------------------------------------------------------------
| pracownicy | CREATE TABLE ‘pracownicy‘ (
  ‘prac_id‘ int(11) NOT NULL auto_increment,
  ‘imie‘ varchar(20) NOT NULL,
  ‘nazwisko‘ varchar(30) NOT NULL,
  ‘pseudo‘ varchar(15) NOT NULL,
  ‘pesel‘ char(11) NOT NULL,
  ‘plec‘ enum(’K’,’M’) NOT NULL,
  ‘platnosci‘ set(’gotowka’,’karta kredytowa’,’przelew’) NOT NULL,
  ‘czy_pracuje‘ char(3) default ’TAK’,
  ‘zarobki‘ decimal(11,2) NOT NULL,
  ‘data_zatr‘ date default NULL,
  PRIMARY KEY  (‘prac_id‘),
  UNIQUE KEY ‘pesel‘ (‘pesel‘),
  UNIQUE KEY ‘imie‘ (‘imie‘,‘nazwisko‘,‘pseudo‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+------------+-------------------------------------------------------------
1 row in set (0.02 sec)
```

2.2. Klucze obce

2.2.1. Podstawowe informacje

Klucze obce (ang. *foreign key*; zwykle w powiązaniu z kluczami głównymi) tworzą mechanizm zabezpieczający przed możliwością powstania niespójności w zapisanych w tabelach danych. Jeżeli dwie tabele powiązane są ze sobą kluczami (tabela, w której jest klucz główny może być traktowana jako nadrzędna a tabela z kluczem obcym jako podrzędna) to nie może zdarzyć się taka sytuacja, że w tabeli podrzędnej będą znajdowały się rekordy, które nie maja „ pasujących rekordów” w tabeli nadrzędnej oraz odwrotnie.

Zagadnienia związane z kluczami obcymi (lub mówiąc ogólniej *integralność* relacyjnych baz danych) została (zostanie) dokładnie omówiona na wykładzie.

Utworzymy więc dwie tabele powiązane ze sobą parą kluczy (główny i obcy). Tabele będą następujące:

Tabela	**Kolumna**	**Ograniczenie**
`studenci`	`stud_id`	PRIMARY KEY
`studenci`	`imie`	NOT NULL
`studenci`	`nazwisko`	NOT NULL
`studenci`	`miasto_id`	FOREIGN KEY
`miasta`	`miasto_id`	PRIMARY KEY
`miasta`	`nazwa`	NOT NULL

Definicja tych tabel w języku SQL (w „dialekcie” MySQL-a) wygląda następująco:

```
DROP TABLE studenci;
DROP TABLE miasta;

CREATE TABLE studenci (
   stud_id      INT              PRIMARY KEY,
   imie         VARCHAR(20)      NOT NULL,
   nazwisko     VARCHAR(30)      NOT NULL,
```

```
   miasto_id    INT
) ENGINE = InnoDB;

CREATE TABLE miasta (
   miasto_id    INT              PRIMARY KEY,
   nazwa        VARCHAR(40)      NOT NULL
) ENGINE = InnoDB;

ALTER TABLE studenci
  ADD FOREIGN KEY (miasto_id) REFERENCES miasta (miasto_id);
```

Ostatnie polecenie SQL tworzy klucz obcy (inaczej: *ograniczenie*) na tabeli *miasta*. Ograniczeniom warto jednak nadawać nazwy, aby łatwiej je było potem zidentyfikować. Gdy jawnie nie nadamy ograniczeniu nazwy zrobi to za nas MySQL. Jednak własne nazwy zwykle są lepsze, bo możemy bardziej dopasować ją do naszych potrzeb (celem np. łatwiejszego zapamiętania). Taka jak poniżej wersja polecenia będzie bardziej „poprawna":

```
ALTER TABLE studenci
  ADD CONSTRAINT studenci_miasto_id_fk FOREIGN KEY (miasto_id)
  REFERENCES miasta (miasto_id);
```

Zwróćmy uwagę, że nazwa ograniczenia została utworzona wg. schematu: *nazwa_tabeli-nazwa_kolumny-fk*. Gdy będziemy konsekwentnie przestrzegać tej konwencji zawsze łatwo z samej nazwy ograniczenia „wyczytamy", o którą tabelę oraz o którą kolumnę chodzi.

Aby przekonać się, że wszystko utworzyło się po naszej myśli możemy wydać poniższe polecenie[1]. W wyświetlonej tabeli widzimy zdefiniowaną przez nas nazwę ograniczenia:

```
SELECT constraint_name, table_schema, table_name, constraint_type
FROM information_schema.table_constraints
WHERE table_schema = 'blab' and table_name='studenci';
```

które powinno dać taki wynik:

```
+-----------------------+--------------+------------+-----------------+
| constraint_name       | table_schema | table_name | constraint_type |
+-----------------------+--------------+------------+-----------------+
| PRIMARY               | blab         | studenci   | PRIMARY KEY     |
| studenci_miasto_id_fk | blab         | studenci   | FOREIGN KEY     |
+-----------------------+--------------+------------+-----------------+
2 rows in set (0.00 sec)
```

Zwróćmy uwagę na pojawiający się w definicjach obu tabel fragment `ENGINE=InnoDB`. W kontekście obsługi kluczy obcych dodanie tej opcji jest obowiązkowe. Co prawda w wersji 5.x serwera MySQL opcja ta jest ustawiana domyślnie, ale nie zaszkodzi (choćby w celach informacyjnych i dokumentacyjnych) ją jawnie podać.

Nie wnikając w tej chwili w zbyt wielkie szczegóły powiedzmy, że MySQL obsługuje kilka różnych rodzajów tabel (dokładniej trzeba by powiedzieć *mechanizmów składowania*). Jak na razie niektóre z nich (styczeń 2006) nie obsługują kluczy obcych, inne natomiast to potrafią. *InnoBD* jest właśnie mechanizmem potrafiącym obsługiwać klucze obce a mechanizm *MyISAM* tego na razie nie potrafi.

[1]Korzystamy tutaj ze specjalnej „systemowej" bazy danych o nazwie *information_schema*. W bazie tej przechowywane są różne informacje na temat innych baz. Jest to więc swego rodzaju *baza metadanych*. Pojawiła się ona dopiero w wersji 5 serwera MySQL. Szczegóły patrz dokumentacja serwera.

Może więc rodzić się pytanie po co utrzymywać w MySQL-u „lepsze” i „gorsze” mechanizmy składowania? Otóż obsługa kluczy obcych (oraz jeszcze kilku innych rzeczy poprawiających niezadowność przechowywania danych — szczególnie istotna jest tutaj obsługa tzw. transakcji) jest po prostu czasochłonna. Gdy więc zależy nam na naprawdę wielkiej wydajności naszej aplikacji powinniśmy rozważyć możliwość przechowywania danych w tabelach *MyISAM*. Gdy absolutnym priorytetem jest bezpieczeństwo, powinniśmy wybrać tabele *InnoDB*.

Ponieważ producent MySQL-a ciągle pracuje nad dodaniem do tabel *MyISAM* obsługi kluczy obcych, więc być może za jakiś czas powyższe zalecenie straci nieco na znaczeniu.

2.2.2. Uzupełnienie

W czasie tworzenia kluczy obcych możemy używać pewnych dodatkowych opcji. Poniżej pokazano fragment składni polecenia `ALTER TABLE` dotyczący tworzenia tych kluczy. Samodzielnie doczytaj w dokumentacji jakie jest znaczenie opcji `RESTRICT`, `CASCADE`, `SET NULL` oraz `NO ACTION`.

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

Klucze obce można również tworzyć w ramach polecenia `CREATE TABLE`. Przykład z poprzedniego podpunktu można więc zapisać w następującej, całkowicie równoważnej postaci:

```
DROP TABLE studenci;
DROP TABLE miasta;

CREATE TABLE miasta (
   miasto_id    INT              PRIMARY KEY,
   nazwa        VARCHAR(40)      NOT NULL
) ENGINE = InnoDB;

CREATE TABLE studenci (
   stud_id      INT              PRIMARY KEY,
   imie         VARCHAR(20)      NOT NULL,
   nazwisko     VARCHAR(30)      NOT NULL,
   miasto_id    INT,
   CONSTRAINT studenci_miasto_id_fk
   FOREIGN KEY (miasto_id)
   REFERENCES miasta (miasto_id)
) ENGINE = InnoDB;
```

Zwróćmy uwagę, że tym razem kolejność tworzenia tabel jest ściśle określona. Najpierw musimy utworzyć tabelę nadrzędną (`miasta`) a dopiero potem tabelę podrzędną (`studenci`). Gdy klucze obce tworzone są z wykorzystaniem polecenie `ALTER TABLE`, kolejność tworzenia tabel jest nieistotna. Oczywistym wymaganiem jest natomiast, aby `ALTER TABLE` pojawiło się *po* poleceniach `CREATE TABLE`.

Po utworzeniu tabel możemy wydać polecenie `SHOW CREATE TABLE`, które potwierdzi nam, że tabela utworzyła się zgodnie z naszymi zamierzeniami:

```
mysql> SHOW CREATE TABLE studenci;
+----------+-------------------------------------------------------------------------------
```

```
| Table    | Create Table
+----------+-------------------------------------------------------------------------------------
| studenci | CREATE TABLE ‘studenci‘ (
  ‘stud_id‘ int(11) NOT NULL,
  ‘imie‘ varchar(20) NOT NULL,
  ‘nazwisko‘ varchar(30) NOT NULL,
  ‘miasto_id‘ int(11) default NULL,
  PRIMARY KEY  (‘stud_id‘),
  KEY ‘studenci_miasto_id_fk‘ (‘miasto_id‘),
  CONSTRAINT ‘studenci_miasto_id_fk‘ FOREIGN KEY (‘miasto_id‘) REFERENCES ‘miasta‘ (‘miasto_id‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+----------+-------------------------------------------------------------------------------------
1 row in set (0.00 sec)

mysql>
```

2.3. Jak MySQL obsługuje ograniczenia

Poniższy tekst został przepisany (z pewnymi skrótami) z rozdziału 1.8.6 z książki [2]. Książka ta jest w zasadzie bardzo wiernym tłumaczeniem większości rozdziałów z [3].

2.3.1. Ograniczenia PRIMARY KEY oraz UNIQUE

MySQL może pracować zarówno z tabelami transakcyjnymi (np. *InnoDB*), które umożliwiają wycofywanie zmian, jak i tabelami nietransakcyjnymi (np. *MyISAM*), które na to nie pozwalają. Z tego powodu obsługa ograniczeń w MySQL jest nieco odmienna niż w innych systemach baz danych (np. *ORACLE*). MySQL musi więc być „przygotowany” na takie sytuacje, gdy aktualizowanych jest wiele wierszy w tabelach nietransakcyjnych, z których *nie można wycofać zmian w razie wystąpienia błędu.*

Kiedy wystąpi błąd, MySQL może zatrzymać wykonywanie instrukcji w środku albo próbować rozwiązać problem i kontynuować pracę.

Próba wstawienia lub aktualizowania wiersza, który powoduje naruszenie ograniczenia `PRIMARY KEY` lub `UNIQUE` zwykle prowadzi do błędu. Gdy używamy tabel transakcyjnych, MySQL automatycznie wycofuje transakcje. W przypadku mechanizmu nietransakcyjnego, MySQL zatrzymuje się przy niepoprawnym wierszu i nie przetwarza pozostałych.

Aby ułatwić pracę programiście, MySQL obsługuje słowo kluczowe `IGNORE` w większości poleceń, które mogą spowodować naruszenie klucza (na przykład `INSERT IGNORE` oraz `UPDATE IGNORE`). W takim przypadku MySQL ignoruje naruszenie klucza i przystępuje do przetwarzania kolejnego wiersza.

2.3.2. Ograniczenia NOT NULL oraz DEFAULT

Aby ułatwić obsługę tabel nietransakcyjnych, w MySQL wszystkie kolumny mają wartości domyślne.

Jeżeli do kolumny zostanie wstawiona „niepoprawna” wartość, na przykład `NULL` do kolumny `NOT NULL` albo zbyt duża wartość do kolumny liczbowej, MySQL, zamiast zgłaszać błąd, ustawia *najlepszą możliwą wartość.*

Zwracamy również uwagę, że od wersji 5.0.2 wprowadzono dwa nowe tryby pracy serwera:

- `STRICT_TRANS_TABLES`,
- `STRICT_ALL_TABLES`.

Pozwalają one wpływać na sposób obsługiwania ‚niepoprawnych" wartości w zależności od tego, czy używamy mechanizmu nietransakcyjnego czy też transakcyjnego.

Zachęcamy do zapoznania się z rozdziałem 1.8.6 pozycji [3] celem poznania szczegółów.

3. Zadanie do samodzielnego wykonania

1. Należy wstawić do tabeli `pracownicy` kilka–kilkanaście przykładowych rekordów. Postarać się celowo naruszać istniejące w tej tabeli ograniczenia i obserwować jak reaguje na to baza MySQL. Przykładowo w kolumnie `platnosc` należy spróbować wstawić wartość nie wymienioną w definicji `set`. Inny przykład: w kolumnie `pesel` spróbować wstawić dwa rekordy z identyczną wartością. Inne przykłady powinieneś opracować samodzielnie.
2. Podobne ćwiczenia należy wykonać dla tabel `studenci` oraz `miasta`.
3. Przećwiczyć działanie opcji `RESTRICT`, `CASCADE`, `SET NULL` oraz `NO ACTION` w poleceniu tworzącym klucze obce.
4. Samodzielnie utworzyć cztery tabele `klienci`, `zamówienia`, `produkty`, `opisy_zamówień` oraz odpowiednio połączyć je kluczami. Jest to bardzo prosta struktura relacyjna (była omawiana na wykładzie) do przechowywania informacji o zamówieniach składanych przez poszczególnych klientów. Rejestrujemy takie informacje jak:
   - dane na temat klientów (doprecyzuj samodzielnie jakie),
   - data złożenia zamówienia oraz kto je złożył,
   - produkty, które pojawiły się na zamówieniu (nazwy, ilość, ewentualnie udzielone rabaty),
   - katalog produktów (nazwy, ceny jednostkowe, opis).

Literatura

[1] Lech Banachowski (tłum.). *SQL. Język relacyjnych baz danych.* WNT Warszawa, 1995.

[2] Paul Dubios. *MySQL. Podręcznik administratora.* Wydawnictwo HELION, 2005.

[3] *MySQL 5.0 Reference Manual*, 2005. (jest to najbardziej aktualne opracowanie na temat bazy MySQL stworzone i na bieżąco aktualizowane przez jej twórców. Książka dostępna w wersji elektronicznej pod adresem *http://dev.mysql.com/doc/*).

[4] Richard Stones and Neil Matthew. *Od podstaw. Bazy danych i MySQL.* Wydawnictwo HELION, 2003.

[5] Luke Welling and Laura Thomson. *MySQL. Podstawy.* Wydawnictwo HELION, 2005.