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

Praca z bazą danych MySQL
wersja 2.0

Nr ćwiczenia:	2
Temat:	**Podstawy języka SQL (Polecenia CREATE, ALTER, DROP, INSERT, UPDATE, DELETE, SELECT)**
Cel ćwiczenia:	Celem ćwiczenia jest zapoznanie studenta z podstawami języka SQL. Pozna on takie polecenia jak: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `CREATE`, `UPDATE`, `DROP`. Większość z omawianych tu zagadnień jest przedstawiona bardzo pobieżnie i zostanie rozwinięta w kolejnych ćwiczeniach.
Wymagane przygotowanie teoretyczne:	—
Sposób zaliczenia:	Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.
Prowadzący zajęcia:	**dr inż. Artur Gramacki, dr inż. Jarosław Gramacki**

1. Uwagi wstępne

1.1. Język SQL

W tym ćwiczeniu poznamy najważniejsze elementy języka SQL (ang. *Structured Query Language*), czyli strukturalnego języka zapytań. Składa się on z dwóch znaczeniowo odrębnych części:

- języka definiowania struktur danych (DDL — ang. *Data Definition Language*), służącego do definiowania (tworzenia, modyfikowania, kasowania) relacyjnych struktur danych (najczęściej będą to tabele, ale też np. widoki, indeksy i inne),
- języka manipulowania danymi (DML — ang. *Data Manipulation Language*), służącego do manipulowania (pobierania, modyfikowania, kasowania) danych zgromadzonych w relacyjnych strukturach danych.

Różni producenci baz danych rozszerzają w różnym stopniu język — mniej lub bardziej zgodnie ze standardem SQL. Przykładowo wymieńmy tutaj polecenia służące do zarządzania transakcjami czy też wykonywania różnych specyficznych zadań administracyjnych. W ćwiczeniu ograniczymy się tylko do pokazania poleceń z grup *DDL* oraz *DML*.

Poznamy (w bardzo wielkim skrócie) następujące polecenia:

Grupa	**Polecenie**	**Opis**
DML	SELECT	wyświetlanie rekordów
DML	INSERT	wstawianie rekordów
DML	UPDATE	modyfikacja rekordów
DML	DELETE	kasowanie rekordów
DDL	CREATE	tworzenie obiektów (np. tabel)
DDL	ALTER	modyfikowanie obiektów (np. tabel)
DDL	DROP	kasowanie obiektów (np. tabel)

1.2. Rozróżnianie wielkości liter

Zanim zaczniemy pisać polecenia w języku SQL oraz tworzyć identyfikatory w MySQL (np. nazwy *baz danych*, nazwy *tabel*, nazwy *kolumn*) powinniśmy omówić kwestię rozróżniania wielkości liter. Otóż standard SQL zakład, że wszelkie słowa kluczowe oraz identyfikatory *nie uwzględniają* wielkości liter. Serwer MySQL również stosuje się do tej zasady, jednak z małym wyjątkiem. Otóż w zależności od systemu operacyjnego pod którym jest zainstalowany serwer, wielkość liter może być istotna w nazwach baz danych oraz nazwach tabel[1]. I tak w systemie UNIX (LINUX) wielkość liter jest istotna a w systemie Windows nie.

Jednak żeby zapewnić przenaszalność danych między różnymi platformami zachęcamy, aby niezależnie od platformy, na której w danym momencie pracujemy zawsze traktować wszystkie identyfikatory tak, jakby rozróżniały wielkość liter.

Wielkości liter nie są natomiast rozróżniane w nazwach kolumn, indeksów i aliasów (dwa ostatnie element zostaną omówione nieco poźniej).

Oczywiście wielkość liter jest rozróżniania w danych, które są zapisywane w tabelach relacyjnych. Przykładowo słowa: *samochód*, *Samochód* oraz *SAMOCHÓD* są w pełni rozróżnialne.

1.3. Studiowanie literatury

Wszystkie omawiane w tym miejscu najważniejsze polecenia języka SQL zostaną przedstawione w *ogromnym skrócie*. Absolutnie nie należy poniższych przykładów traktować jako kompendium wiedzy na tema języka SQL. Przeciwnie, zachęcamy do samodzielnego studiowania dostępnej literatury (zwłaszcza polecamy pozycję [3]). Celem ćwiczenia jest bowiem pokazanie jedynie najistotniejszych elementów SQL-a. Aby unaocznić Ci, jak bardzo pobieżnie traktujemy w ćwiczeniu poszczególne polecenia SQL poniżej przytaczamy kompletny diagram syntaktyczny polecenia `SELECT`. W ćwiczeniach, które będziesz wykonywał pewne bardziej szczegółowe (i mniej istotne z praktycznego punktu widzenia) elementy nie zostaną wcale omówione.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr, ...
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name']
    [FROM table_references
    [WHERE where_definition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_definition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC] , ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

[1]Utworzenie nowej bazy danych skutkuje utworzeniem nowego katalogu dyskowego o takiej samej nazwie, jak nazwa nowotworzonej bazy. Z kolei utworzenie tabeli skutkuje powstaniem na dysku pliku o takiej nazwie, jak nazw tabeli.

2. Polecenie CREATE TABLE

Aby rozpocząć ćwiczenie należy zalogować się oraz wybrać bazę danych, do której mamy prawa dostępu (patrz poprzednia instrukcja).

Poniższe polecenie utworzy nam bardzo prostą tabelę (relację) o nazwie *studenci*.

```
/*==============================================================*/
/* Tabela: studenci                                             */
/*==============================================================*/
-- To też jest znak komentarza (wszystko za znakami '--'
-- jest pomijane przez analizator SQL-a
DROP TABLE IF EXISTS studenci;

CREATE TABLE studenci
(
   stud_id        INT            NOT NULL AUTO_INCREMENT PRIMARY KEY,
   imie           VARCHAR(20)    NOT NULL,
   nazwisko       VARCHAR(30)    NOT NULL,
   typ_uczel_id   CHAR(1)            NULL
);
```

Uwagi:

- stosujemy konsekwentnie zasadę, że słowa kluczowe języka SQL piszemy wielkimi literami a identyfikatory (np. *imie*, *nazwisko*, *studenci*) małymi. Zachęcamy do konsekwentnego stosowania tej konwencji, gdyż poprawia to czytelność kodów,
- całe polecenie może być wpisywane „ciurkiem”[2], jednak warto konsekwentnie stosować wcięcia oraz przejścia do nowej linii (podobnie jak poprzednio, zabiegi te poprawiają czytelność kodów),
- tekst, który występuje między znakami `/*` oraz `*/` jest traktowany jako komentarz i pomijany przez interpreter języka SQL. Innym rodzajem komentarza są dwa znaki minus, często stosowane, aby skomentować niewielką ilość tekstu (wszystko za tymi znakami jest pomijane przez analizator SQL-a),
- nasza bardzo prosta pierwsza tabela (*relacja*) składa się z czterech kolumn: pierwsza jest typu całkowitego (kolumna `stud_id`), kolejne dwie są typu znakowego o zmiennej szerokości (kolumny `imie` oraz `nazwisko`), ostatnia kolumna (`typ_uczel_id`) jest też typu znakowego, ale o stałej szerokości,
- kolumna `stud_id` jest tzw. kluczem głównym (będzie o tym mowa na wykładzie) oraz jest zdefiniowana jako `AUTO_INCREMENT`. Wartość tej kolumny przy dodawaniu rekordów jest automatycznie zwiększana. Gdy dodajemy wiersz do tabeli zawierającej kolumnę typu `AUTO_INCREMENT`, nie podajemy wartości dla tej kolumny (pleceniem `INSERT`), bo odpowiedni numer nada baza danych. Pozwala to na proste i efektywne rozwiązanie problemu generowania unikatowych wartości dla kolejnych wierszy tabeli.
- system MySQL obsługuje wiele różnych typów danych. Dokumentacja bardzo dokładnie to opisuje, więc nie będziemy w tym miejscu podawać tych informacji. Wszystkie je można jednak podzielić na następujące podstawowe grupy. W ramach każdej z grup wymieniono najważniejsze warianty. Szczegóły doczytaj w dokumentacji:
  - typ *logiczny* (`TRUE`, `FALSE`, `NULL`),

[2] Sam przekonaj się, czy takie polecenie będzie czytelne: `CREATE TABLE studenci (stud_id INT NOT NULL,imie VARCHAR(20) NOT NULL,nazwisko VARCHAR(30) NOT NULL,typ_uczel_id CHAR(1))`?

  - typy *liczbowe* (INTEGER (oraz jego odmiany), FLOAT, DOUBLE, NUMERIC),
  - związane *z datą i czasem* (DATE, TIME, DATETIME, TIMESTAMP, YEAR, DATETIME),
  - typy *łańcuchowe* (CHAR, VARCHAR, TEXT (oraz jego odmiany), ENUM, SET),
  - *duże obiekty binarne* (BLOB).
- trzy pierwsze kolumny mają status NOT NULL, co oznacza, że dla każdego wiersza w tej tabeli w tych kolumnach *musi być obowiązkowo wpisana jakaś wartość* (nie można pozostawić wartości pustych),
- całość polecenia zakończona jest średnikiem.

Aby przekonać się, czy tabela rzeczywiście jest w systemie wykonajmy następujące polecenie:

```
mysql> show tables;
+----------------+
| Tables_in_blab |
+----------------+
| studenci       |
+----------------+
1 row in set (0.00 sec)
```

Więcej informacji na temat struktury każdej tabeli można uzyskać, używając pokazanej niżej instrukcji. Czy potrafisz właściwie odczytać i zrozumieć wyświetloną tabelkę?

```
mysql> describe studenci;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| stud_id      | int(11)     | NO   | PRI | NULL    | auto_increment |
| imie         | varchar(20) | NO   |     |         |                |
| nazwisko     | varchar(30) | NO   |     |         |                |
| typ_uczel_id | char(1)     | YES  |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
4 rows in set (0.06 sec)
```

2.1. Polecenie CREATE TABLE ... AS SELECT

Ciekawą możliwością jest tworzenie tabel w oparciu o inne, istniejące już tabele. Możliwość taka przydaje się wówczas, gdy np. w szybki sposób chcemy utworzyć kopię istniejącej tabeli (gdy np. planujemy eksperymentowanie z pewną tabelą i chcemy się zabezpieczyć przed omyłkowym uszkodzeniem danych). Zobaczmy jak wygląda to w praktyce:

```
CREATE TABLE studenci_2
AS SELECT
  imie, nazwisko, typ_uczel_id
FROM
  studenci
WHERE
  typ_uczel_id = 'P';

mysql> SELECT * FROM studenci_2;
+-------+------------+--------------+
| imie  | nazwisko   | typ_uczel_id |
+-------+------------+--------------+
```

```
| Artur | Nowakowski | P            |
| Jan   | Kowalski   | P            |
| Marek | Pawlak     | P            |
+-------+------------+--------------+
3 rows in set (0.00 sec)
```

Nowotworzona tabela może być oparta o dane niekoniecznie z jednej tylko tabeli. W klauzuli `AS SELECT` możemy wpisywać dowolnie złożone zapytanie SQL. W przykładzie poniżej pobieramy dane z dwóch różnych tabel i łączymy je w jedną wynikową tabelę.

Najpierw tworzymy tabelę *uczelnie*, która będzie na potrzebna do demonstracji oraz zapełniamy ją przykładowymi danymi. Zakładamy ponadto, że tabela *studenci* też zawiera już kilka przykładowych rekordów. Dane do tabel wstawiamy poleceniem `INSERT` – szczegóły patrz rozdział 3.

```
CREATE TABLE uczelnie
(
   typ_uczel_id    CHAR(1)        NOT NULL PRIMARY KEY,
   nazwa           VARCHAR(20)    NOT NULL
);

INSERT INTO uczelnie VALUES ('U', 'Uniwersytet');
INSERT INTO uczelnie VALUES ('P', 'Politechnika');
INSERT INTO uczelnie VALUES ('A', 'Akademia');
```

Teraz już możemy utworzyć nową tabelę:

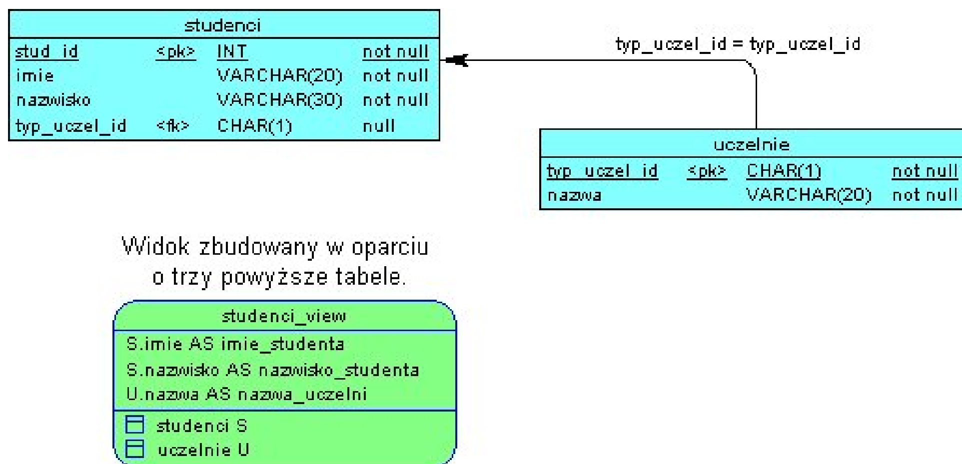
```
CREATE TABLE studenci_3
AS SELECT
  S.imie, S.nazwisko, U.nazwa
FROM
  studenci AS S, uczelnie AS U
WHERE
  U.typ_uczel_id = S.typ_uczel_id AND
  U.nazwa = 'Politechnika';

mysql>  SELECT * FROM studenci_3;
+-------+------------+--------------+
| imie  | nazwisko   | nazwa        |
+-------+------------+--------------+
| Artur | Nowakowski | Politechnika |
| Jan   | Kowalski   | Politechnika |
| Marek | Pawlak     | Politechnika |
+-------+------------+--------------+
3 rows in set (0.00 sec)
```

2.2. Polecenie CREATE VIEW

Widoki[3] (ang. *view*) zostały zaimplementowane w wersji 5. serwera MySQL. Są to obiekty bardzo przydatne i funkcjonalne. W poprzednim podpunkcie tworzyliśmy nowe tabele w oparciu o tabele już istniejące. Nowotworzone tabele oczywiście zajmują określoną ilość miejsca na dysku i często tworzenie nowych tabel w celach innych niż jako np. kopie bezpieczeństwa jest dość dyskusyjne.

Możliwe jest jednak utworzenie tzw. widoku, który może być traktowany jako bardzo wygodny sposób prezentacji danych w wymaganym przez użytkownika układzie. Gdy przykładowo bardzo

[3]Widoki czasami nazywane są też *perspektywami*.



Rysunek 1: Przykładowy widok

często musimy wyświetlać dane o studentach (tabela *studenci*) w połączeniu z danymi z innych tabel (np. *uczelnie*), to wygodniej jest utworzyć stosowny widok, niż za każdym razem wpisywać dość długie polecenie SQL. Na rysunku 1 pokazano szczegóły omawianego widoku.

Zwróćmy uwagę, że utworzenie widoku nie powoduje skopiowania danych, które używane są w widoku. Serwer MySQL przechowuje tylko definicję widoku a dane pobierane są *on-line* w momencie odwołania do niego. Mamy więc do czynienia z sytuacją zupełnie inną w przypadku polecenia `CREATE TABLE ... AS SELECT`.

Polecenie tworzące omawiany widok wygląda następująco:

```
CREATE OR REPLACE VIEW studenci_view AS
SELECT
  S.imie AS imie_studenta,
  S.nazwisko AS nazwisko_studenta,
  U.nazwa AS nazwa_uczelni
FROM
  studenci S, uczelnie U
WHERE
  S.typ_uczel_id = U.typ_uczel_id;
```

Następnie z widoku korzystamy w bardzo podobny sposób jak z tabeli. Zwróćmy uwagę, że nazwy kolumn w widoku są inne niż w tabelach bazowych. Oczywiście nazwy te mogą równie dobrze brzmieć tak samo. Decyzja należy do Ciebie. Nie ma to obsolutnie żadnego wpływu na działanie widoku.

```
mysql> SELECT * FROM studenci_view;
+---------------+-------------------+---------------+
| imie_studenta | nazwisko_studenta | nazwa_uczelni |
+---------------+-------------------+---------------+
| Artur         | Nowakowski        | Politechnika  |
| Jan           | Kowalski          | Politechnika  |
| Roman         | Nowak             | Uniwersytet   |
| Stefan        | Antkowiak         | Akademia      |
| Ewa           | Konieczna         | Akademia      |
| Anna          | Wojtasik          | Akademia      |
| Marek         | Pawlak            | Politechnika  |
+---------------+-------------------+---------------+
7 rows in set (0.01 sec)
```

Korzystanie z widoku wiąże się jednak z pewnymi ograniczeniami. Przykładowo nie uda się operacja wykasowania danych z widoku, gdy widok ten jest zbudowany w oparciu o więcej niż jedną tabelę. Podobne ograniczenia wystąpią przy próbie wstawiania rekordów. Z powyższego widać, że ograniczeń w używaniu widoków jest dużo i w praktyce używane są one raczej tylko w trybie do odczytu. Więcej szczegółów na temat pracy z widokami znajdziesz w literaturze.

```
mysql> DELETE FROM studenci_view;
ERROR 1395 (HY000): Can not delete from join view 'blab.studenci_view'


mysql> INSERT INTO studenci_view VALUES
    -> ('Artur', 'Gramacki', 'Politechnika');
ERROR 1394 (HY000): Can not insert into join view 'blab.studenci_view' without
                    fields list

mysql> INSERT INTO studenci_view (imie_studenta, nazwisko_studenta, nazwa_uczelni)
    -> VALUES ('Artur', 'Gramacki', 'Politechnika');
ERROR 1393 (HY000): Can not modify more than one base table through a join view
                    'blab.studenci_view'
mysql>
```

3. Polecenie INSERT

Obecnie załadujemy do naszej tabeli kilka przykładowych rekordów:

```
INSERT INTO studenci VALUES (1, 'Artur', 'Nowakowski', 'P');
INSERT INTO studenci VALUES (2, 'Jan', 'Kowalski', 'P');
INSERT INTO studenci VALUES (3, 'Roman', 'Nowak', 'U');
INSERT INTO studenci VALUES (4, 'Stefan', 'Antkowiak', 'A');
INSERT INTO studenci VALUES (5, 'Ewa', 'Konieczna', 'A');
INSERT INTO studenci VALUES (6, 'Anna', 'Wojtasik', 'A');
INSERT INTO studenci VALUES (7, 'Marek', 'Pawlak', 'P');
COMMIT;
```

Uwagi:

- podobnie jak przy poleceniu `CREATE`, polecenie `INSERT` również musi być zakończone średnikiem,
- ostatnie polecenie (`COMMIT`) zatwierdza wprowadzone w tabeli zmiany. Nie wnikając w tej chwili w szczegóły powiedzmy tylko tyle, że zatwierdzanie jest konieczne w tzw. *tabelach transakcyjnych* (np. *InnoDB*) a w *tabelach nietransakcyjnych* (np. *MyISAM*) nie ma zastosowania,
- dane alfanumeryczne muszą być ujęte z apostrofy lub cydzysłowy. Pola numeryczne nie muszą być ujmowane w apostrofy, choć gdy będą ujęte to nic się złego nie stanie. MySQL po prostu dokona automatycznej konwersji do pola numerycznego.

W systemie MySQL można również stosować nieco bardziej zwartą postać polecenie `INSERT`. Nie jest to co prawda zgodne ze standardem SQL ale bardzo upraszcza zapis i jest w związku z tym chętnie stosowane:

```
INSERT INTO studenci VALUES
  (1, 'Artur', 'Nowakowski', 'P'),
  (2, 'Jan', 'Kowalski', 'P'),
```

```
(3, 'Roman', 'Nowak', 'U'),
(4, 'Stefan', 'Antkowiak', 'A'),
(5, 'Ewa', 'Konieczna', 'A'),
(6, 'Anna', 'Wojtasik', 'A'),
(7, 'Marek', 'Pawlak', 'P');
```

Pamiętamy, że przy tworzeniu tabeli *studenci* jedna z kolumn została zdefiniowana jako typ AUTO_INCREMENT. Aby wykorzystać możliwość automatycznego generowania unikalnych wartości dla tego typu kolumny musimy użyć innej wersji polecenia INSERT, która pozwoli nam na wstawianie danych tylko do wskazanych kolumn:

```
INSERT INTO studenci (imie, nazwisko) VALUES ('Artur', 'Gramacki') ;
```

Zwróćmy również uwagę, że nie podaliśmy wartości dla kolumny typ_uczel_id. Mogliśmy tak uczynić, gdyż kolumna ta jest zdefiniowana jako NULL, czyli może nie być tam wpisana żadna wartość.

Na koniec zobaczmy, jakie dane są w tabeli *studenci*. Łatwo zauważamy, że mimo iż nie podaliśmy jawnie wartości 8 dla kolumny typ_uczel_id, system i tak wpisał nam poprawną wartość:

```
+---------+--------+------------+--------------+
| stud_id | imie   | nazwisko   | typ_uczel_id |
+---------+--------+------------+--------------+
|       1 | Artur  | Nowakowski | P            |
|       2 | Jan    | Kowalski   | P            |
|       3 | Roman  | Nowak      | U            |
|       4 | Stefan | Antkowiak  | A            |
|       5 | Ewa    | Konieczna  | A            |
|       6 | Anna   | Wojtasik   | A            |
|       7 | Marek  | Pawlak     | P            |
|       8 | Artur  | Gramacki   | NULL         |
+---------+--------+------------+--------------+
8 rows in set (0.00 sec)
```

Polecenie INSERT można też używać w taki sposób jak pokazano poniżej. Wymieniamy po prostu nazwy kolejnych kolumn i zaraz potem podajemy wartości dla tych kolumn. Podana tutaj forma instrukcji INSERT jest w pełni równoważna z tą podaną powyżej. Możesz więc wybrać sobie tą wersję, która Ci najbardziej odpowiada.

```
INSERT INTO studenci SET imie='Tomasz', nazwisko='Wisniewski';
```

3.1. Polecenie INSERT ... SELECT

W MySQL istnieje bardzo ciekawa opcja polecenia SELECT. Można mianowicie wstawić dane do tabeli pobierając je z innej tabeli. Zobaczmy jak wygląda to w praktyce. Najpierw tworzymy nową tabelę:

```
mysql> CREATE TABLE studenci_P (imie VARCHAR(25), nazwisko VARCHAR(25));
Query OK, 0 rows affected (0.10 sec)
```

Następnie do nowoutworzonej tabeli wstawiamy wybrane dane z „bazowej” tabeli *studenci*:

```
INSERT INTO studenci_P (imie, nazwisko)
SELECT imie, nazwisko
FROM studenci
WHERE typ_uczel_id = 'P';
```

Na koniec sprawdzamy zawartość tabeli *studenci_P*:

```
mysql> SELECT * FROM studenci_P;
+-------+------------+
| imie  | nazwisko   |
+-------+------------+
| Artur | Nowakowski |
| Jan   | Kowalski   |
| Marek | Pawlak     |
+-------+------------+
3 rows in set (0.00 sec)
```

4. Polecenie SELECT

Obecnie zajmiemy się jednym z najczęściej wykonywanych poleceń w relacyjnych bazach danych. Służy ono do pobierania (np. wyświetlania na ekranie) danych zapisanych w tabelach (lub tzw. *widokach*; ang. *view*). Poniżej pokazujemy (omawiane na wykładzie) polecenia *selekcji* oraz *projekcji*. Nie omawiamy ich szczegółowo, więc postaraj się (być może posiłkując się dokumentacją) zrozumieć ich sens i stosowaną składnię. W kolejnych ćwiczeniach będziemy jeszcze wracać do tych zagadnień.

```
mysql> SELECT * FROM studenci;
+---------+--------+------------+--------------+
| stud_id | imie   | nazwisko   | typ_uczel_id |
+---------+--------+------------+--------------+
|       1 | Artur  | Nowakowski | P            |
|       2 | Jan    | Kowalski   | P            |
|       3 | Roman  | Nowak      | U            |
|       4 | Stefan | Antkowiak  | A            |
|       5 | Ewa    | Konieczna  | A            |
|       6 | Anna   | Wojtasik   | A            |
|       7 | Marek  | Pawlak     | P            |
+---------+--------+------------+--------------+
7 rows in set (0.00 sec)
```

```
mysql> SELECT nazwisko, imie, typ_uczel_id, stud_id FROM studenci;
+------------+--------+--------------+---------+
| nazwisko   | imie   | typ_uczel_id | stud_id |
+------------+--------+--------------+---------+
| Nowakowski | Artur  | P            |       1 |
| Kowalski   | Jan    | P            |       2 |
| Nowak      | Roman  | U            |       3 |
| Antkowiak  | Stefan | A            |       4 |
| Konieczna  | Ewa    | A            |       5 |
| Wojtasik   | Anna   | A            |       6 |
| Pawlak     | Marek  | P            |       7 |
+------------+--------+--------------+---------+
7 rows in set (0.00 sec)
```

```
mysql> SELECT nazwisko, imie FROM studenci;
+------------+--------+
| nazwisko   | imie   |
+------------+--------+
| Nowakowski | Artur  |
| Kowalski   | Jan    |
| Nowak      | Roman  |
| Antkowiak  | Stefan |
| Konieczna  | Ewa    |
| Wojtasik   | Anna   |
| Pawlak     | Marek  |
+------------+--------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM studenci WHERE stud_id > 3;
+---------+--------+-----------+--------------+
| stud_id | imie   | nazwisko  | typ_uczel_id |
+---------+--------+-----------+--------------+
|       4 | Stefan | Antkowiak | A            |
|       5 | Ewa    | Konieczna | A            |
|       6 | Anna   | Wojtasik  | A            |
|       7 | Marek  | Pawlak    | P            |
+---------+--------+-----------+--------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM studenci LIMIT 3, 4;
+---------+--------+-----------+--------------+
| stud_id | imie   | nazwisko  | typ_uczel_id |
+---------+--------+-----------+--------------+
|       4 | Stefan | Antkowiak | A            |
|       5 | Ewa    | Konieczna | A            |
|       6 | Anna   | Wojtasik  | A            |
|       7 | Marek  | Pawlak    | P            |
+---------+--------+-----------+--------------+
4 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM studenci;
+----------+
| COUNT(*) |
+----------+
|        7 |
+----------+
1 row in set (0.04 sec)
```

4.1. Pewne ciekawe możliwości formatowania wyniku

Czasami wygodniej jest oglądać rezultaty zapytania w pionie a nie w tradycyjnym układzie poziomej tabeli (są wówczas czytelniejsze). Aby osiągnąć taki efekt, należy zakończyć polecenie nie znakiem średnika ale sekwencją `\G`. Porównajmy:

```
mysql> SELECT * FROM studenci \G
*************************** 1. row ***************************
     stud_id: 1
        imie: Artur
    nazwisko: Nowakowski
typ_uczel_id: P
```

```
************************** 2. row ***************************
     stud_id: 2
        imie: Jan
    nazwisko: Kowalski
typ_uczel_id: P

...

************************** 7. row ***************************
     stud_id: 7
        imie: Marek
    nazwisko: Pawlak
typ_uczel_id: P
7 rows in set (0.00 sec)
```

Ciekawą cechą MySQL-a jest możliwość wyświetlania wyników w postaci kodów `HTML` oraz `XML`. Aby „zmusić" do tego program klienta *mysql* należy uruchomić go z opcją `--html` lub `XML` (lub użyć skróconych wersji odpowiednio `-H` oraz `-X`). Porównajmy:

```
shell> mysql -u ulab -phlab --html blab

mysql> SELECT * FROM studenci;
<TABLE BORDER=1><TR><TH>stud_id</TH><TH>imie</TH><TH>nazwisko</TH><TH>typ_uczel_
id</TH></TR><TR><TD>1</TD><TD>Artur</TD><TD>Nowakowski</TD><TD>P</TD></TR><TR><T
D>2</TD><TD>Jan</TD><TD>Kowalski</TD><TD>P</TD></TR><TR><TD>3</TD><TD>Roman</TD>
<TD>Nowak</TD><TD>U</TD></TR><TR><TD>4</TD><TD>Stefan</TD><TD>Antkowiak</TD><TD>
A</TD></TR><TR><TD>5</TD><TD>Ewa</TD><TD>Konieczna</TD><TD>A</TD></TR><TR><TD>6<
/TD><TD>Anna</TD><TD>Wojtasik</TD><TD>A</TD></TR><TR><TD>7</TD><TD>Marek</TD><TD
>Pawlak</TD><TD>P</TD></TR></TABLE>7 rows in set (0.00 sec)
```

Wynik jest co prawda sformatowany bardzo nieczytelnie (nie ma żadnych wcięć ani przejść do nowej linii) jednak można go stosunkowo łatwo „wyczyścić". Niewątpliwie jednak możliwość otrzymania wyniku, który można natychmiast wyświetlić w przeglądarce WWW jest bardzo ciekawa. Na szczęście przy formacie `XML` wynik jest sformatowany bardziej czytelnie:

```
shell> mysql -u ulab -phlab --xml blab

mysql> SELECT * FROM studenci;
<?xml version="1.0"?>

<resultset statement="SELECT * FROM studenci">
  <row>
        <field name="stud_id">1</field>
        <field name="imie">Artur</field>
        <field name="nazwisko">Nowakowski</field>
        <field name="typ_uczel_id">P</field>
  </row>

  <row>
        <field name="stud_id">2</field>
        <field name="imie">Jan</field>
        <field name="nazwisko">Kowalski</field>
        <field name="typ_uczel_id">P</field>
  </row>

...
```

```
  <row>
        <field name="stud_id">7</field>
        <field name="imie">Marek</field>
        <field name="nazwisko">Pawlak</field>
        <field name="typ_uczel_id">P</field>
  </row>
</resultset>
7 rows in set (0.00 sec)
```

5. Polecenie UPDATE

Wstawione do tabel dane nie są zwykle w pełni statyczne. Czasami trzeba je po prostu zmodyfikować (zmienić). Modyfikowanie danych wykonujemy za pomocą polecenie `UPDATE`. Podobnie jak przy poleceniu `SELECT` nie omawiamy go zbyt dokładnie. Choć trzeba przyznać, że polecenie to jest bardzo proste i łatwe do samodzielnego opanowania. Czy potrafisz się domyśleć, co wykonuje poniższe polecenie?

```
mysql> UPDATE studenci SET nazwisko='Kowalska', imie='Edyta'
    -> WHERE stud_id=2;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Sprawdzamy, czy się udało:

```
mysql> SELECT * FROM studenci WHERE stud_id=2;
+---------+-------+----------+--------------+
| stud_id | imie  | nazwisko | typ_uczel_id |
+---------+-------+----------+--------------+
|       2 | Edyta | Kowalska | P            |
+---------+-------+----------+--------------+
1 row in set (0.03 sec)
```

W czasie modyfikowania danych trzeba bardzo uważać. Pominięcie bowiem klauzuli `WHERE` może nieoczekiwanie zmodyfikować wszystkie rekordy w tabeli. Przeanalizujmy kolejny przykład:

```
mysql> UPDATE studenci SET nazwisko='Kowalska', imie='Edyta';
Query OK, 7 rows affected (0.11 sec)
Rows matched: 7  Changed: 7  Warnings: 0
```

Następnie zobaczmy, co otrzymaliśmy. Taki wynik prawdopodobnie rzadko kiedy jest wynikiem pożądanym. Gdy omyłkowo na trwale zatwierdzimy wprowadzone zmiany[4], oryginalne dane będzie trudno odzyskać (lub nawet może się zdarzyć, że odzyskanie danych w ogóle nie będzie możliwe!).

```
mysql> SELECT * FROM studenci;
+---------+-------+----------+--------------+
| stud_id | imie  | nazwisko | typ_uczel_id |
+---------+-------+----------+--------------+
```

[4]Uwaga: MySQL domyślnie jest tak konfigurowany, że wprowadzane zmiany są *natychmiast* trwale zapisywane na dysku. Gdy używamy tzw. *mechanizmu składowania InnoDB* można to zachowanie zmienić. Gdy natomiast używamy mechanizmu o nazwie *MyISAM* nie mamy wyboru — zmiany są natychmiast trwale zapisywane.

```
|       1 | Edyta | Kowalska | P            |
|       2 | Edyta | Kowalska | P            |
|       3 | Edyta | Kowalska | U            |
|       4 | Edyta | Kowalska | A            |
|       5 | Edyta | Kowalska | A            |
|       6 | Edyta | Kowalska | A            |
|       7 | Edyta | Kowalska | P            |
+---------+-------+----------+--------------+
7 rows in set (0.00 sec)
```

6. Polecenie DELETE oraz TRUNCATE

Polecenie do kasowania rekordów (`DELETE`) jest bardzo podobne do polecenie `UPDATE`. Wiążą się z nim te same niebezpieczeństwa. Tym razem utraty a nie zmodyfikowania danych — czasami nie wiadomo co jest lepsze `:-(`.

```
mysql> DELETE FROM studenci WHERE stud_id > 3;
Query OK, 4 rows affected (0.03 sec)

mysql> SELECT * FROM studenci;
+---------+-------+------------+--------------+
| stud_id | imie  | nazwisko   | typ_uczel_id |
+---------+-------+------------+--------------+
|       1 | Artur | Nowakowski | P            |
|       2 | Jan   | Kowalski   | P            |
|       3 | Roman | Nowak      | U            |
+---------+-------+------------+--------------+
3 rows in set (0.00 sec)
```

Pominięcie klauzuli `WHERE` może nieoczekiwanie wykasować wszystkie rekordy w tabeli. Taki wynik prawdopodobnie rzadko kiedy jest wynikiem pożądanym. Gdy omyłkowo na trwale zatwierdzimy wprowadzone zmiany, oryginalne dane będzie trudno odzyskać (lub nawet może się zdarzyć, że odzyskanie danych w ogóle nie będzie możliwe!):

```
mysql> DELETE FROM studenci;
Query OK, 3 rows affected (0.03 sec)

mysql> SELECT * FROM studenci;
Empty set (0.00 sec)
```

Gdy chcemy szybko usunąć *wszystkie* dane z tabeli możemy użyć polecenie `TRUNCATE`, które jest dużo szybsze w działaniu od polecenia `DELETE`. Zmianę w szybkości działania jestśmy jednak w stanie zauważyć dopiero wówczas, gdy tabela zawiera naprawdę dużo rekordów (rzędu tysięcy a nawet więcej). Wzrost szybkości uzyskujemy dlatego, że tabela jest najpierw po prostu kasowana a następnie tworzona na nowo (już pusta).

```
TRUNCATE TABLE studenci;
```

6.1. „Bezpieczne” uruchamianie serwera

Konsolę MySQL można uruchomić w specjalnym „bezpiecznym” trybie, z użyciem przełącznika:

`--safe-updates`,

który powoduje, że nie jest możliwe wykonywanie poleceń UPDATE oraz DELETE bez klauzuli WHERE. W czasie działania konsoli SQL można również wydać polecenie:

SET SQL_SAFE_UPDATES = 1,

co da ten sam efekt. Porównajmy:

```
mysql> DELETE FROM studenci;
ERROR 1175 (HY000): You are using safe update mode and you tried to
update a tab le without a WHERE that uses a KEY column mysql>
```

7. Polecenie ALTER TABLE

Czasami zachodzi potrzeba zmodyfikowania struktury tabeli. Oczywiście najlepiej byłoby, gdyby już na etapie projektowania aplikacji bazodanowej można było ustalić definitywnie jej strukturę relacyjną. Oczywiście jest to sytuacja idealna, jednak życie projektanta pokazuje, że wcześniej czy później (byle nie za często) należy wprowadzić w strukturze tabel jakieś zmiany. Wymagane zmiany należy oczywiście wprowadzić w taki sposób, aby nie utracić istniejących już danych. Poniższe polecenie:

- usuwa z tabeli studenci kolumnę `typ_uczel_id`,
- zmienia nazwę kolumny z `stud_id` na `student_id`,
- dodaje dwie nowe kolumny: `data_urodzenia` oraz `plec`.

```
ALTER TABLE studenci DROP COLUMN typ_uczel_id;
ALTER TABLE studenci CHANGE COLUMN stud_id student_id INT;
ALTER TABLE studenci ADD COLUMN data_urodzenia DATE;
ALTER TABLE studenci ADD COLUMN plec CHAR(1);
```

Po dokonaniu zmian wyświetlamy nową definicję tabeli, co potwierdza nam, że zmiany zostały dokonane właściwie:

```
mysql> DESCRIBE studenci;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| student_id     | int(11)     | YES  |     | NULL    |       |
| imie           | varchar(20) | NO   |     |         |       |
| nazwisko       | varchar(30) | NO   |     |         |       |
| data_urodzenia | date        | YES  |     | NULL    |       |
| plec           | char(1)     | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Na koniec zobaczmy jak wyglądają dane w zmienionej tabeli. Widzimy, że MySQL wpisał wartość NULL (co należy rozumieć mniej więcej jako „wartość nieznana w tym momencie”):

```
mysql> SELECT * FROM studenci;
+------------+-------+------------+----------------+------+
| student_id | imie  | nazwisko   | data_urodzenia | plec |
+------------+-------+------------+----------------+------+
|          1 | Artur | Nowakowski | NULL           | NULL |
|          2 | Jan   | Kowalski   | NULL           | NULL |
```

```
|          3 | Roman  | Nowak      | NULL             | NULL |
|          4 | Stefan | Antkowiak  | NULL             | NULL |
|          5 | Ewa    | Konieczna  | NULL             | NULL |
|          6 | Anna   | Wojtasik   | NULL             | NULL |
|          7 | Marek  | Pawlak     | NULL             | NULL |
+------------+--------+------------+-----------------+------+
7 rows in set (0.00 sec)
```

W kolejnym kroku samodzielnie zmodyfikuj zawartość tabeli *studenci* (polecenie `UPDATE`) tak, aby zamiast wartości `NULL` w poszczególnych rekordach pojawiły się jakieś dane. Przy wpisywaniu dat pamiętaj, że w MySQL-u domyślnym formatem dla daty jest `YYYY-MM-DD`. W kolumnie `plec` dopuszczalnymi wartościami niech będą `M` oraz `K`.

W kolumnie `data_urodzenia` celowo wpisuj daty w niewłaściwym formacie i obserwuj jakim błędami odpowiada serwer. Podobne eksperymenty wykonaj dla innych kolumn. Przykładowo „zapomnij" o wpisaniu apostrofów przy podawaniu wartości dla pól znakowych lub też w kolumnie numerycznej wpisuj ciągi znakowe. Postaraj się nabrać biegłości we właściwym odczytywaniu komunikatów systemowych. Pozwala to zaoszczędzić sporo czasu i nerwów!

8. Polecenie DROP TABLE

Ostatnie omawiane polecenie służy do kasowania całych tabel. Oczywiście skasowanie tabeli jest równoznaczne ze skasowaniem wszystkich znajdujących się w niej rekordów. Polecenie to należy więc wydawać bardzo ostrożnie !

```
mysql> DROP TABLE studenci;
Query OK, 0 rows affected (0.04 sec)
```

Potwierdzamy, że tabela rzeczywiście już nie istnieje:

```
mysql> SELECT * FROM studenci;
ERROR 1146 (42S02): Table 'blab.studenci' doesn't exist
```

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.