W tej części zajmiemy się ćwiczeniami dotyczącymi modyfikacji rekordów.

Logujemy się do bazy danych (jak pamiętamy, służy do tego oprogramowanie klienta, czyli programik `mysql`).

Załóżmy sobie przede wszystkim nową bazę do ćwiczeń. Zanim to zrobimy warto upewnić się, jakie bazy już istnieją. Wykonajmy polecenie SHOW DATABASES.

UWAGA: Polecenie SHOW DATABASES **nie jest** poleceniem języka SQL! Polecenie to jest charakterystyczne dla bazy danych mysql (choć większość serwerów obsługuje podobne narzędzia).

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| poligon            |
| test               |
+--------------------+
4 rows in set (0.02 sec)
```

Mamy, jak widać 3 standardowe bazy danych mysql i naszą bazę poligon. Usuńmy bazę poligon i załóżmy ją ponownie (dzięki czemu będzie ona na pewno pusta). Przy okazji zobaczymy, że w jednej linijce można wydawać kilka poleceń SQL – wystarczy oddzielać je średnikiem.

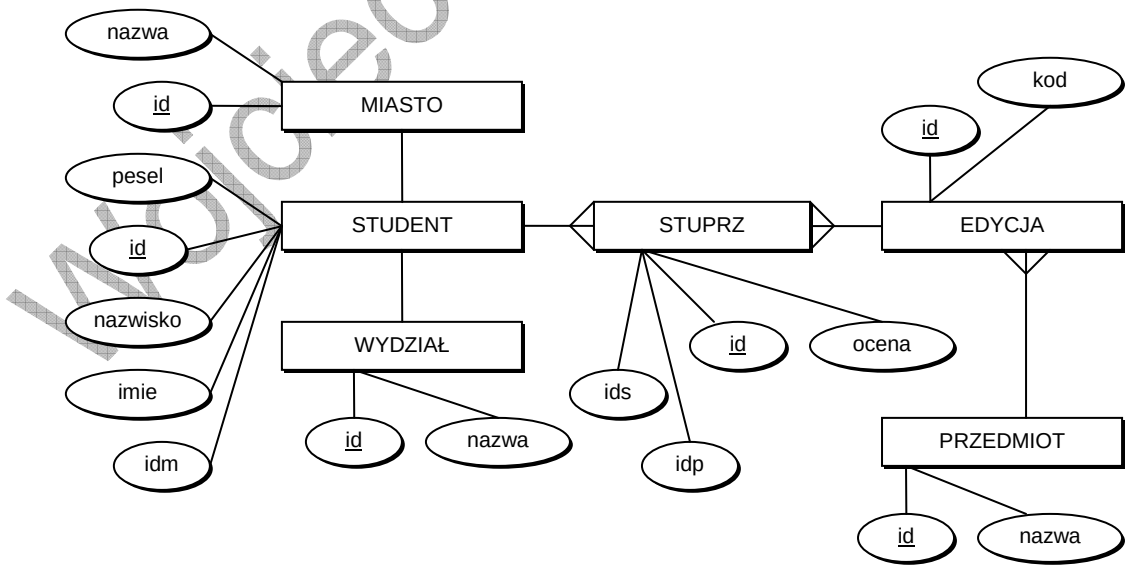
```
mysql> drop database poligon; create database poligon;
Query OK, 0 rows affected (0.05 sec)

Query OK, 1 row affected (0.01 sec)
```

Teraz trzeba podłączyć się do bazy poligon (prawdopodobnie nadal jesteśmy jeszcze w bazie głównej, czyli mysql). Robimy to za pomocą polecenia USE (to także jest komenda specyficzna dla mysql). Alternatywnie, możemy wyjść z klienta mysql (służy do tego skrót /q) i zalogować się z parametrem –D poligon.

```
mysql> USE poligon;
Database changed
```

Załóżmy sobie tablice bazy danych na podstawie takiego oto diagramu ERD:




```
mysql> create table miasto (id int auto_increment primary key, nazwa char(50));
mysql> create table wydzial (id int auto_increment primary key, nazwa char(50));
mysql> create table student (id int auto_increment primary key, nazwa char(50),
imie char(20), nazwisko char(50), pesel char(11));
```

```
mysql> create table stuprz (id int auto_increment primary key, ids int, idp int)
mysql> create table przedmiot (id int auto_increment primary key, nazwa char(50)
);
mysql> create table edycja (id int auto_increment primary key, kod char(10));
```

Sprawdźmy, czy utworzyły się wszystkie tablice:

```
mysql> show tables;
+-------------------+
| Tables_in_poligon |
+-------------------+
| edycja            |
| miasto            |
| przedmiot         |
| student           |
| stuprz            |
| wydzial           |
+-------------------+
```

Wnikliwy czytelnik zauważy, że tablica STUDENT nie została utworzona w takiej formie, jak widnieje na diagramie ERD. Zajmiemy się tym jednak później.

Wprowadźmy kilka miast. Jak pamiętamy, służy do tego polecenie INSERT INTO. Ma ono następującą składnię (poniższy zapis nie obejmuje wszystkich możliwości):

INSERT INTO nazwa_tabeli (kolumna) VALUES (wartość)

Pamiętać należy, że z zasady nie ustawia się wartości pól oznaczonych jako auto_increment. Teoretycznie można to zrobić (o ile wartości się nie powtórzą), ale nie jest to zgodne z regułami dobrego programowania.

W tym miejscu warto też przypomnieć, że własność kolumny auto_increment nie jest częścią standardu SQL (czy raczej: SQL92). W oryginale funkcjonowało pojęcie SEQUENCE (sekwencji), czyli specjalnego obiektu, którego wielkość zwiększała się o 1 przy każdym odczycie, co pozwalało korzystać z niego jako z generatora zwiększających się liczb.

```
mysql> insert into miasto (nazwa) values ('Gdansk');
Query OK, 1 row affected (0.13 sec)

mysql> insert into miasto (nazwa) values ('Poznan');
Query OK, 1 row affected (0.01 sec)

mysql> insert into miasto (nazwa) values ('Krakow');
Query OK, 1 row affected (0.03 sec)

mysql> insert into miasto (nazwa) values ('Lodz');
Query OK, 1 row affected (0.05 sec)

mysql> insert into miasto (nazwa) values ('Warszawa');
Query OK, 1 row affected (0.00 sec)

mysql> insert into miasto (nazwa) values ('Szczecin');
Query OK, 1 row affected (0.03 sec)

mysql> insert into miasto (nazwa) values ('Bialystok');
Query OK, 1 row affected (0.02 sec)
```

Jeśli tabela ma więcej niż jedną kolumnę, posługujemy się taką postacią polecenia INSERT:

INSERT INTO nazwa_tabeli (kol1, kol2, ... kolN) VALUES (wartość1, wartość1, wartośćN)

```
mysql> insert into student (imie, nazwisko) values ('Iwona', 'Rybicka');
Query OK, 1 row affected (0.02 sec)

mysql> insert into student (imie, nazwisko) values ('Karol', 'Stefanski');
Query OK, 1 row affected (0.02 sec)

mysql> insert into student (imie, nazwisko) values ('Jakub', 'Malinowski');
Query OK, 1 row affected (0.02 sec)

mysql> insert into student (imie, nazwisko) values ('Jerzy', 'Kaczorowski');
Query OK, 1 row affected (0.03 sec)
```

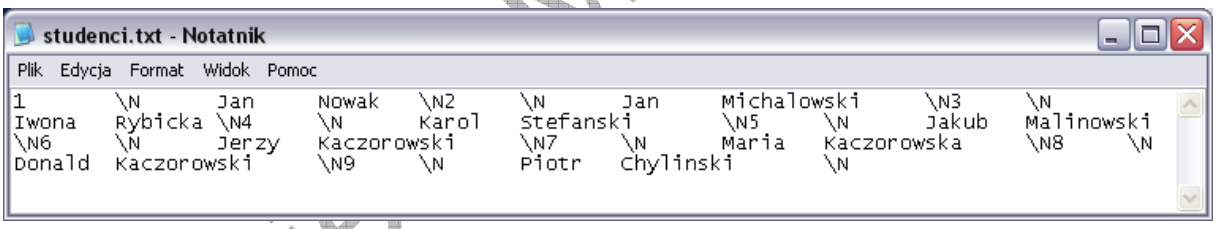
W ten sposób możemy wpisać jeszcze kilka rekordów. Sprawdźmy, jakie wartości wpisaliśmy do tabeli:

```
mysql> SELECT * FROM student;
+----+-------+--------+-------------+-------+
| id | nazwa | imie   | nazwisko    | pesel |
+----+-------+--------+-------------+-------+
|  1 | NULL  | Jan    | Nowak       | NULL  |
|  2 | NULL  | Jan    | Michalowski | NULL  |
|  3 | NULL  | Iwona  | Rybicka     | NULL  |
|  4 | NULL  | Karol  | Stefanski   | NULL  |
|  5 | NULL  | Jakub  | Malinowski  | NULL  |
|  6 | NULL  | Jerzy  | Kaczorowski | NULL  |
|  7 | NULL  | Maria  | Kaczorowska | NULL  |
|  8 | NULL  | Donald | Kaczorowski | NULL  |
|  9 | NULL  | Piotr  | Chylinski   | NULL  |
+----+-------+--------+-------------+-------+
9 rows in set (0.02 sec)
```

Warto zwrócić uwagę na interesujące polecenie mysql, które umożliwia przesłanie wyniku zapytania SELECT do pliku. Służy do tego konstrukcja SELECT (...) INTO OUTFILE nazwa_pliku:

```
mysql> SELECT * FROM student INTO OUTFILE 'c:\studenci.txt';
Query OK, 9 rows affected (0.02 sec)
```

Po jego wykonaniu, utworzony zostaje plik o następującej zawartości:



Jak widać, jest to bardzo prosty plik tekstowy – co jest akurat w typ przypadku zaletą, bo umożliwia w miarę bezbolesne przenoszenie pliku pomiędzy platformami takimi jak Unix i Windows. W tym miejscu powiedzmy też, że aby „importować" pliki danych do mysql powinny mieć one właśnie taką „prostą" postać. Oznacza to, że żeby zaimportować plik np. Excela (czy nawet inny tekstowy – w którym pola oddzielają np. przecinki) musimy pobawić się trochę z jego edycją.

Usuńmy teraz istniejące rekordy z tablicy i sprawdźmy, czy udało się to zrobić:

```
mysql> delete from student;
Query OK, 9 rows affected (0.33 sec)

mysql> select * from student;
Empty set (0.00 sec)
```

Do ładowania danych z pliku służy polecenie LOAD DATA INFILE plik INTO TABLE tablica:

```
mysql> LOAD DATA INFILE 'c:\studenci.txt' INTO TABLE student;
Query OK, 9 rows affected (0.06 sec)
Records: 9  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from student;
+----+-------+--------+-------------+-------+
| id | nazwa | imie   | nazwisko    | pesel |
+----+-------+--------+-------------+-------+
|  1 | NULL  | Jan    | Nowak       | NULL  |
|  2 | NULL  | Jan    | Michalowski | NULL  |
|  3 | NULL  | Iwona  | Rybicka     | NULL  |
|  4 | NULL  | Karol  | Stefanski   | NULL  |
|  5 | NULL  | Jakub  | Malinowski  | NULL  |
|  6 | NULL  | Jerzy  | Kaczorowski | NULL  |
|  7 | NULL  | Maria  | Kaczorowska | NULL  |
|  8 | NULL  | Donald | Kaczorowski | NULL  |
|  9 | NULL  | Piotr  | Chylinski   | NULL  |
+----+-------+--------+-------------+-------+
9 rows in set (0.00 sec)
```

Jak widać, w tablicy STUDENT znajduje się niepotrzebna kolumna nazwa. Z drugiej strony, brakuje pola idm łączącego ją z tabelą MIASTO. Ponieważ nie chcemy utracić wpisanych rekordów, dokonamy modyfikacji działającej tabeli, do czego posłużymy się poleceniem ALTER TABLE:

Po pierwsze usuniemy istniejącą, zbędną kolumnę:

ALTER TABLE student DROP COLUMN nazwa

```
mysql> ALTER TABLE student DROP COLUMN nazwa;
Query OK, 9 rows affected (0.44 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> select * from student;
+----+--------+-------------+-------+
| id | imie   | nazwisko    | pesel |
+----+--------+-------------+-------+
|  1 | Jan    | Nowak       | NULL  |
|  2 | Jan    | Michalowski | NULL  |
|  3 | Iwona  | Rybicka     | NULL  |
|  4 | Karol  | Stefanski   | NULL  |
|  5 | Jakub  | Malinowski  | NULL  |
|  6 | Jerzy  | Kaczorowski | NULL  |
|  7 | Maria  | Kaczorowska | NULL  |
|  8 | Donald | Kaczorowski | NULL  |
|  9 | Piotr  | Chylinski   | NULL  |
+----+--------+-------------+-------+
9 rows in set (0.00 sec)
```

Po drugie dostawiamy nową kolumnę (oczywiście specyfikując jej typ):

ALTER TABLE student ADD COLUMN idm int

```
mysql> ALTER TABLE student ADD COLUMN idm int;
Query OK, 9 rows affected (0.23 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> select * from student;
+----+--------+-------------+-------+------+
| id | imie   | nazwisko    | pesel | idm  |
+----+--------+-------------+-------+------+
|  1 | Jan    | Nowak       | NULL  | NULL |
|  2 | Jan    | Michalowski | NULL  | NULL |
|  3 | Iwona  | Rybicka     | NULL  | NULL |
|  4 | Karol  | Stefanski   | NULL  | NULL |
|  5 | Jakub  | Malinowski  | NULL  | NULL |
|  6 | Jerzy  | Kaczorowski | NULL  | NULL |
|  7 | Maria  | Kaczorowska | NULL  | NULL |
|  8 | Donald | Kaczorowski | NULL  | NULL |
|  9 | Piotr  | Chylinski   | NULL  | NULL |
+----+--------+-------------+-------+------+
9 rows in set (0.00 sec)
```

Jak widać, polecenie ALTER TABLE służy do wykonywania operacji na całej tabeli (ściślej: na jej strukturze).

Kolumna PESEL zawiera wartości puste. Żeby być ścisłym – są to wartości NULL, czyli: nie ustawione, bowiem formalnie wartością pustą kolumny typu char jest łańcuch pusty: ‘’.
Spróbujmy ustawić je na wartość „brak danych”. Służy do tego polecenie UPDATE:

UPDATE student SET pesel = ‘brak danych’

```
mysql> UPDATE student SET pesel = 'brak danych';
Query OK, 9 rows affected (0.00 sec)
Rows matched: 9  Changed: 9  Warnings: 0

mysql> select * from student;
+----+--------+-------------+-------------+------+
| id | imie   | nazwisko    | pesel       | idm  |
+----+--------+-------------+-------------+------+
|  1 | Jan    | Nowak       | brak danych | NULL |
|  2 | Jan    | Michalowski | brak danych | NULL |
|  3 | Iwona  | Rybicka     | brak danych | NULL |
|  4 | Karol  | Stefanski   | brak danych | NULL |
|  5 | Jakub  | Malinowski  | brak danych | NULL |
|  6 | Jerzy  | Kaczorowski | brak danych | NULL |
|  7 | Maria  | Kaczorowska | brak danych | NULL |
|  8 | Donald | Kaczorowski | brak danych | NULL |
|  9 | Piotr  | Chylinski   | brak danych | NULL |
+----+--------+-------------+-------------+------+
9 rows in set (0.00 sec)
```

Jak widać, UPDATE zmieniło wartości wszystkich rekordów. Polecenie to może jednak być zastosowane dla wybranych wierszy tabeli:

```
mysql> UPDATE student SET pesel = '23548801783' WHERE id=4;
Query OK, 1 row affected (0.22 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student;
+----+--------+-------------+-------------+------+
| id | imie   | nazwisko    | pesel       | idm  |
+----+--------+-------------+-------------+------+
|  1 | Jan    | Nowak       | brak danych | NULL |
|  2 | Jan    | Michalowski | brak danych | NULL |
|  3 | Iwona  | Rybicka     | brak danych | NULL |
|  4 | Karol  | Stefanski   | 23548801783 | NULL |
|  5 | Jakub  | Malinowski  | brak danych | NULL |
|  6 | Jerzy  | Kaczorowski | brak danych | NULL |
|  7 | Maria  | Kaczorowska | brak danych | NULL |
|  8 | Donald | Kaczorowski | brak danych | NULL |
|  9 | Piotr  | Chylinski   | brak danych | NULL |
+----+--------+-------------+-------------+------+
9 rows in set (0.00 sec)
```

Spróbujmy teraz zasymulować błędne działanie – ustawmy ten sam pesel innej osobie.

```
mysql> UPDATE student SET pesel = '23548801783' WHERE id=5;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Jak widać, serwer nie protestuje. Dzieje się tak dlatego, że kolumna pesel nie ma wartości UNIQUE (unikalność).

Spróbujmy jeszcze raz skorzystać z polecenia ALTER TABLE. Jest ono na tyle wszechstronne, że umożliwia także najróżniejsze modyfikacje kolumn – łącznie ze zmianą typu. Warto zwrócić tu uwagę, że polecenie ALTER TABLE składa się niejako z dwóch części: po pierwszej określamy tabelę, na której chcemy dokonać zmian, a następnie precyzujemy, co chcemy osiągnąć (np. ADD COLUMN, DROP COLUMN, albo MODIFY COLUMN):

ALTER TABLE student MODIFY COLUMN pesel char(11) UNIQUE

```
mysql> ALTER TABLE student MODIFY COLUMN pesel char(11) UNIQUE;
ERROR 1062 (23000): Duplicate entry 'brak danych' for key 2
```

Oczywiście baza danych napotyka na problem związany z wielokrotnym wpisem ‘brak danych’ – co kłóci się z postulatem unikalności. Jedyną wartością, która może się powtarzać w takiej kolumnie jest NULL:

```
mysql> UPDATE student set pesel=NULL;
Query OK, 2 rows affected (0.03 sec)
Rows matched: 9  Changed: 2  Warnings: 0

mysql> ALTER TABLE student MODIFY COLUMN pesel char(11) UNIQUE;
Query OK, 9 rows affected (0.09 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

Sprawdźmy teraz, co stałoby się, gdybyśmy próbowali wprowadzić ponownie ten sam pesel:

```
mysql> UPDATE student SET pesel = '23548801783' WHERE id=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE student SET pesel = '23548801783' WHERE id=5;
ERROR 1062 (23000): Duplicate entry '23548801783' for key 2
```

Jak widać, ochroniliśmy naszą bazę danych przed błędnymi wpisami. Nadal jednak nie mamy ustawionych połączeń z tablicą MIASTO (zawiera ono miasto, w którym urodził się nasz student). Pokażemy teraz prostą sztuczkę, która pozwoli nam uporać się z tym problemem. Po pierwsze, sprawdźmy, jakie wartości id występują w tablicy MIASTO:

```
mysql> select * from miasto;
+----+-----------+
| id | nazwa     |
+----+-----------+
|  1 | Gdansk    |
|  2 | Poznan    |
|  3 | Krakow    |
|  4 | Lodz      |
|  5 | Warszawa  |
|  6 | Szczecin  |
|  7 | Bialystok |
+----+-----------+
```

Jak widać, są to wartości od 1 do 7. Oznacza to, że powinniśmy do pola idm tabeli STUDENT wpisywać właśnie takie liczby. Sztuczka polega na wykorzystaniu funkcji RAND.

Serwer bazy danych ma wbudowany generator liczb losowych – jest nim funkcja rand(). Możemy wywołać ją za pomocą SELECT:

```
mysql> SELECT rand() from DUAL;
+------------------+
| rand()           |
+------------------+
| 0.42458005940968 |
+------------------+
```

Uważny czytelnik zapewne zauważy, że coś jest niejasne. SELECT ... FROM powinno odnosić się do tablicy, podczas gdy nie utworzyliśmy tablicy o nazwie dual. Nie zauważmy jej także, jeśli wykonamy SHOW TABLES:

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_poligon |
+-------------------+
| edycja            |
| miasto            |
| przedmiot         |
| student           |
| stuprz            |
| wydzial           |
+-------------------+
6 rows in set (0.00 sec)
```

Zapytanie odnoszące się do funkcji rand() zwróciło jednak poprawny wynik. Dzieje się tak dlatego, że każda baza SQL ma wbudowaną „niewidoczną” tabelę o nazwie dual. Jej istnienie było potrzebne

właśnie po to, aby można było wykonywać takie pobrania wartości, jak to widzieliśmy w przykładzie z random. Co ciekawe, funkcja agregująca count() pokazuje, że w tabeli dual znajduje się jeden rekord, ale nie można go wyświetlić zwykłym selectem:

```
mysql> SELECT count(*) from DUAL;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT * from DUAL;
ERROR 1096 (HY000): No tables used
```

Warto też dodać, że mysql pozwala na ominięcie frazy FROM w tym przypadku – ale nie każda baza danych zachowuje się w ten sposób. Innymi słowy, w przypadku serwera mysql polecenia
SELECT rand() FROM dual
SELECT rand()
są równoznaczne.

Funkcja rand() zwraca wartości z przedziału <0,1). Ponieważ interesuje nas większy zakres zmienności, jej wynik pomnożymy przez 6. Oznacza to, że 6*rand() jest liczbą z zakresu <0,6). Teraz wystarczy dodać 1, aby uzyskać przedział <1,7).

```
mysql> SELECT 6*rand()+1;
+------------------+
| 6*rand()+1       |
+------------------+
| 4.4510862486913 |
+------------------+
1 row in set (0.00 sec)
```

Ostatnim krokiem jest zaokrąglenie wyniku, co zapewnia nam, że otrzymamy liczbę całkowitą z przedziału 1-7:

```
mysql> SELECT round(6*rand()+1);
+--------------------+
| round(6*rand()+1) |
+--------------------+
|                  4 |
+--------------------+
1 row in set (0.01 sec)
```

Pozostaje teraz wykorzystać zgromadzona wiedzę do automatycznego (choć LOSOWEGO) przypisania miast do studentów. Taka operacja nie ma sensu w rzeczywistej bazie, ale jest często wykorzystywana do prac testowych lub dla inicjalnego wypełnienia danymi.

```
mysql> UPDATE student SET idm = round(6*rand()+1);
Query OK, 7 rows affected (0.01 sec)
Rows matched: 9  Changed: 7  Warnings: 0

mysql> select * from student;
+----+--------+-------------+-------------+------+
| id | imie   | nazwisko    | pesel       | idm  |
+----+--------+-------------+-------------+------+
|  1 | Jan    | Nowak       | NULL        |    3 |
|  2 | Jan    | Michalowski | NULL        |    6 |
|  3 | Iwona  | Rybicka     | NULL        |    2 |
|  4 | Karol  | Stefanski   | 23548801783 |    2 |
|  5 | Jakub  | Malinowski  | NULL        |    3 |
|  6 | Jerzy  | Kaczorowski | NULL        |    3 |
|  7 | Maria  | Kaczorowska | NULL        |    4 |
|  8 | Donald | Kaczorowski | NULL        |    5 |
|  9 | Piotr  | Chylinski   | NULL        |    2 |
+----+--------+-------------+-------------+------+
9 rows in set (0.00 sec)
```

Dla ćwiczenia, można sprawdzić, czy pamiętamy, jak policzyć liczbę studentów w każdym mieście:

```
mysql> select nazwa, count(student.id) from miasto JOIN STUDENT ON (miasto.id=st
udent.idm) GROUP BY nazwa;
+----------+-------------------+
| nazwa    | count(student.id) |
+----------+-------------------+
| Krakow   |                 3 |
| Lodz     |                 1 |
| Poznan   |                 3 |
| Szczecin |                 1 |
| Warszawa |                 1 |
+----------+-------------------+
```

Usuńmy teraz jednego studenta – na przykład tego, któremu wpisaliśmy już pesel.

Możemy to zrobić na dwa sposoby – albo specyfikując określone id, albo nakładając warunek na kolumnę pesel:

DELETE FROM student WHERE id = 4
Lub
DELETE FROM student WHERE pesel IS NOT NULL

```
mysql> DELETE FROM student where pesel IS NOT NULL;
Query OK, 1 row affected (0.05 sec)

mysql> select * from student;
+----+--------+-------------+-------+------+
| id | imie   | nazwisko    | pesel | idm  |
+----+--------+-------------+-------+------+
|  1 | Jan    | Nowak       | NULL  |    3 |
|  2 | Jan    | Michalowski | NULL  |    6 |
|  3 | Iwona  | Rybicka     | NULL  |    2 |
|  5 | Jakub  | Malinowski  | NULL  |    3 |
|  6 | Jerzy  | Kaczorowski | NULL  |    3 |
|  7 | Maria  | Kaczorowska | NULL  |    4 |
|  8 | Donald | Kaczorowski | NULL  |    5 |
|  9 | Piotr  | Cylinski    | NULL  |    2 |
+----+--------+-------------+-------+------+
8 rows in set (0.00 sec)
```

W tym miejscu jedna uwaga – raczej należy unikać intuicyjnej składni pesel <> NULL. Większość serwerów baz danych traktuje wartość NULL specyficznie i dlatego dla jej „obsługi” w warunkach logicznych używa się klauzul: kolumna IS NULL lun kolumna IS NOT NULL.