Bazy danych

5. SQL- podstawy

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2020

SQL - *Structured Query Language*

Interpretowany język programowania. Jego polecenia tradycyjnie nazywane są *zapytaniami* (kwerendami, ang. *query*).

- front-end relacyjnych baz danych
- back-end aplikacji korzystających z relacyjnych baz danych ☺

W sieci dostępnych jest **bardzo** dużo kursów i tutoriali SQL.

Używane standardy

- SQL-86
- SQL-92
- SQL:1999
- **SQL:2003**
- SQL:2006
- SQL:2008
- SQL:2011
- SQL:2016

Żadna z komercyjnych lub otwartych implementacji SQL nie jest w pełni zgodna ze statandardem (ponad SQL:1999), chociaż wszystkie są *prawie* zgodne ze standardem SQL:2003. Dostarczyciele wprowadzają za to własne rozszerzenia języka, tak więc skrypty SQL i aplikacje korzystające z SQL nie są w pełni przenoszalne.

Na tej stronie `http://troels.arvin.dk/db/rdbms/` wymienone są różnice pomiędzy standardem SQL a jego popularnymi implementacjami.

“Części” SQL

- DDL (*Data Definition Language*) — zmiana schematu bazy danych, tworzenie, modyfikacja i usuwanie tabel, indeksów itd (`CREATE`, `ALTER`, `DROP`).
- DML (*Data Manipulation Language*) — wprowadzanie, modyfikacja, usuwanie danych (`INSERT`, `UPDATE`, `DELETE`).
- DQL (*Data Query Language*), niekiedy traktowany jako część DML — właściwy język zapytań (`SELECT` i wszystko, co się z nim łączy).
- DCL (*Data Control Language*) — nadawanie uprawnień do obiektów bazodanowych (`GRANT`, `REVOKE`, `DENY`).
- rozszerzenia o funkcje i procedury składowane, w tym kursory.

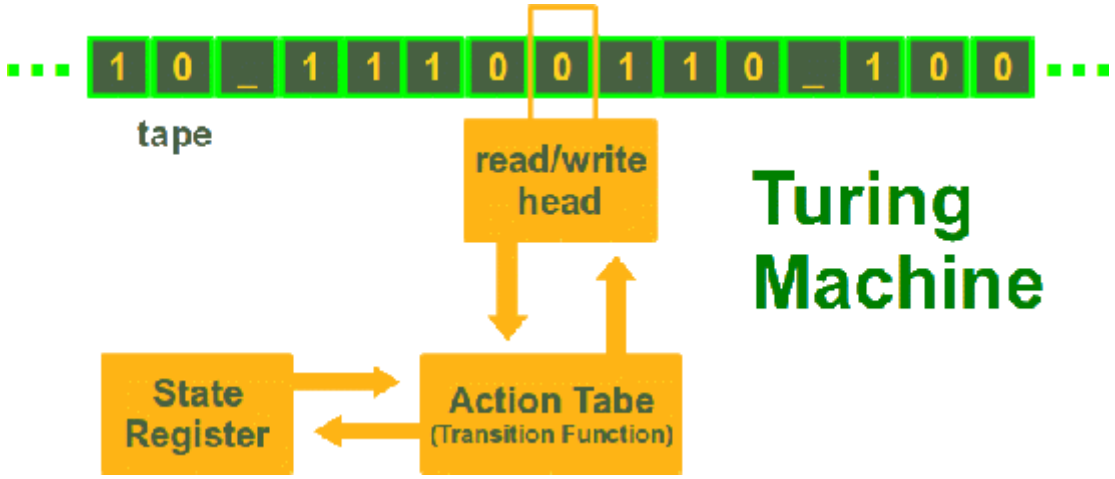
Programowanie deklaratywne

SQL — a ściślej, DQL — jest językiem *deklaratywnym*: określa logikę programowania, bez określenia *control flow*. Innymi słowy, zapytanie SQL mówi **co** chcemy osiągnąć, bez określenia **jak** chcemy to zrobić.

Przeciwieństwem deklaratywnego paradygmatu programowania jest programowanie *imperatywne*.

Czy SQL jest *językiem programowania*?

Jeśli ograniczymy się do DDL, DML, DQL i DCL, SQL **nie** jest językiem programowania, gdyż nie jest zupełny w sensie Turinga. Taki “klasyczny” SQL jeswt jedynie językiem manipulowania danymi. Jeśli jednak uwzględnimy rozszerzenia SQL w postaci procedur i funkcji składowanych, SQL **jest** językiem programowania.



Dygresja: Zupełność w sensie Turinga (*Turing completeness*)

Zespół reguł manipulowania danymi jest zupełny w sensie Turinga jeśli można za jego pomocą zasymulować uniwersalną maszynę Turinga.

Mówimy o “regułach manipulowania danymi”, gdyż maszyna Turinga w istocie to robi: odczytuje i zapisuje symbole z taśmy wejścia-wyjścia.

Korzystając z pojęcia równoważności w sensie Turinga, możemy powiedzieć, że za pomocą SQL rozszerzonego o procedury i funkcje składowane, można zasymulowaćdziałanie każdego innego systemu (programu) komputerowego zupełnego w sensie Turinga — abstrahując od wygody, efektywności i innych praktycznych aspektów takiej symulacji.

Połączenie się z serwerem

Przed rozpoczęciem pracy, nalezy wywołać proces kliencki, który połaczy się z serwerem. W MySQL może to wyglądać na przykład tak:

```
C:\>mysql -upfg -p
Enter password: ************
```

Po zakończeniu pracy trzeba się pożegnać:

```
mysql> QUIT;
Bye
```

Na ogół połączenie z serwerem nawiązuje się za pomocą jakiejś aplikacji, niekiedy za pośrednictwem pliku wsadowego, nie zaś bezpośrednio z shella.

SELECT

Najczęściej używanym zapytaniem SQL jest `SELECT`. Służy ono do uzyskiwania informacji o zawartości tabel, ale może służyć także do wielu innych celów, na przykład do obliczeń matematycznych.

```
mysql> SELECT 2+2;
+-----+
| 2+2 |
+-----+
|   4 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT Sin(0.25*Pi()),Log(10.0),Log2(16.0);
+--------------------+-------------------+------------+
| Sin(0.25*Pi())     | Log(10.0)         | Log2(16.0) |
+--------------------+-------------------+------------+
| 0.7071067811865476 | 2.302585092994046 |          4 |
+--------------------+-------------------+------------+
1 row in set (0.00 sec)
```

Symbol NULL w wyrażeniach

W SQL istnieje specjalna wartość, NULL, na oznaczenie brakujących danych. Jeśli NULL stanie się elementem obliczeń matematycznych lub logicznych, ich wynikiem jest NULL.

```
mysql> SELECT 2+NULL, TRUE AND NULL, FALSE OR NULL;
+--------+---------------+---------------+
| 2+NULL | TRUE AND NULL | FALSE OR NULL |
+--------+---------------+---------------+
|   NULL |          NULL |          NULL |
+--------+---------------+---------------+
1 row in set (0.05 sec)
```

Ale (co jest niezgodne ze standardem!)

```
mysql> SELECT NULL OR TRUE;
+--------------+
| NULL OR TRUE |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)
```

Zmienne tymczasowe

SQL pozwala na definiowanie zmiennych tymczasowych — nazwa zmiennej tymczasowej zawsze zaczyna się od `@`. Zmienna istnieje dopóty, dopóki nie zostanie zakończone połączenie z serwerem. Procesy klienckie nie widzą zmiennych zdefiniowanych przez *inne* procesy.

```
mysql> SET @a=5;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT @a-2;
+------+
| @a-2 |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

Zmiennej można przypisać wartość, która jest *wynikiem zapytania*:

```
mysql> SET @b=(SELECT NrStudenta FROM Studenci2 WHERE Imie='Tadeusz');
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @b, @a+@b;
+------+-------+
| @b   | @a+@b |
+------+-------+
| 74   | 79    |
+------+-------+
1 row in set (0.00 sec)
```

Utworzenie bazy danych

```
mysql> CREATE DATABASE MoiStudenci;
Query OK, 1 row affected (0.06 sec)
```

Uwaga na systemy znaków! Jeśli chcemy używac polskich znaków diakrytycznych, niekiedy należy *explicite* określić `CHARACTER SET`. To silnie zależy od systemu operacyjnego, implementacji DBMS, a nawet jej wersji.

```
mysql> USE MoiStudenci;
Database changed
mysql>
```

Po wywołaniu klienta, trzeba “wejść” do wybranej bazy za pomocą instrukcji `USE` — nie tylko po jej utworzeniu, ale zawsze.

To, jakie bazy znajdują się na serwerze, możemy zobaczyć za pomocą instrukcji `SHOW DATABASES`.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kaskady            |
| moistudenci        |
| mysql              |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

Utworzenie tabeli

```
mysql> CREATE TABLE Studenci
    -> (NrStudenta SMALLINT UNSIGNED NOT NULL
    ->  AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.14 sec)
```

Jeśli okaże się to konieczne, po utworzeniu można zmienić definicję tabeli.

```
mysql> ALTER TABLE Studenci
    -> ADD COLUMN (Nazwisko VARCHAR(20) NOT NULL);
Query OK, 1 row affected (0.17 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

Składnia polecenia `CREATE TABLE`

`CREATE TABLE` NazwaTabeli (
NazwaKolumny$_1$ TypKolumny$_1$ `NULL` | `NOT NULL` ...,
NazwaKolumny$_2$ TypKolumny$_2$ `NULL` | `NOT NULL` ...,
...
NazwaKolumny$_n$ TypKolumny$_n$ `NULL` | `NOT NULL` ...,
`PRIMARY KEY` (NazwaKolumny$_p$,NazwaKolumny$_q$,...)
)
;

Inne opcje poznamy później.

Typy danych (atrybutów)

TINYINT	DATE	TINYBLOB
SMALLINT	TIME	BLOB
MEDIUMINT	TIMESTAMP	MEDIUMBLOB
INT	DATETIME	LONGBLOB
INTEGER	CHAR	TINYTEXT
BIGINT	VARCHAR	TEXT
REAL	BOOLEAN	LONGTEXT
DOUBLE		ENUM
FLOAT		SET
DECIMAL		
NUMERIC		

Instrukcja `DESCRIBE` podaje definicję tabeli.

```
mysql> DESCRIBE Studenci;
+------------+----------------------+------+-----+---------+----------------+
| Field      | Type                 | Null | Key | Default | Extra          |
+------------+----------------------+------+-----+---------+----------------+
| NrStudenta | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| Nazwisko   | varchar(20)          | NO   |     |         |                |
+------------+----------------------+------+-----+---------+----------------+
2 rows in set (0.01 sec)
```

Niepotrzebną tabelę usuwamy za pomocą instrukcji `DROP TABLE`.

```
mysql> DROP TABLE Studenci;
Query OK, 0 rows affected (0.02 sec)
```

Lepiej ją utworzyć od początku dobrze niż dodawać kolumna po kolumnie. . . ☺

```
mysql> CREATE TABLE Studenci
    -> (NrStudenta SMALLINT UNSIGNED NOT NUll
    ->  AUTO_INCREMENT PRIMARY KEY,
    ->  Imie VARCHAR(20),
    ->  Nazwisko VARCHAR(20) NOT NULL,
    ->  Uwagi VARCHAR(30),
    ->  Grupa CHAR(2) NOT NULL);
Query OK, 0 rows affected (0.06 sec)
mysql> DESCRIBE Studenci;
+------------+---------------------+------+-----+---------+----------------+
| Field      | Type                | Null | Key | Default | Extra          |
+------------+---------------------+------+-----+---------+----------------+
| NrStudenta | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| Imie       | varchar(20)         | YES  |     | NULL    |                |
| Nazwisko   | varchar(20)         | NO   |     |         |                |
| Uwagi      | varchar(30)         | YES  |     | NULL    |                |
| Grupa      | char(2)             | NO   |     |         |                |
+------------+---------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Wstawianie wartości do tabeli

```
mysql> INSERT INTO Studenci VALUES
    -> (1,'Alicja','Zielińska','','pn');
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM Studenci;
+------------+--------+-----------+-------+-------+
| NrStudenta | Imie   | Nazwisko  | Uwagi | Grupa |
+------------+--------+-----------+-------+-------+
|          1 | Alicja | Zielińska |       | pn    |
+------------+--------+-----------+-------+-------+
1 row in set (0.15 sec)
```

Składnia zapytania `SELECT`;

```
SELECT [DISTINCT] lista_select
FROM tabela_lub_zapytanie
WHERE warunek_logiczny
GROUP BY wyrażenie_grupujące
HAVING warunek_wyszukiwania_po_grupowaniu
ORDER BY wyrażenie_porządkujące [ASC | DESC]
;
```

W zapytaniu `SELECT` *wszystkie klauzule muszą występować w powyższej kolejności.* Niektóre klauzule można opuścić, ale to, co jest, musi być w tej kolejności.

Składnia zapytania `SELECT`;

```
SELECT [DISTINCT] lista_select
FROM tabela_lub_zapytanie
WHERE warunek_logiczny
GROUP BY wyrażenie_grupujące
HAVING warunek_wyszukiwania_po_grupowaniu
ORDER BY wyrażenie_porządkujące [ASC | DESC]
;
```

W zapytaniu `SELECT` *wszystkie klauzule muszą występować w powyższej kolejności.* Niektóre klauzule można opuścić, ale to, co jest, musi być w tej kolejności.

Dodajmy więcej danych

```
mysql> INSERT INTO Studenci VALUES
    -> (2,'Bogdan','Żółkiewski','','pn'),
    -> (3,'Czesław','Ygrekowski','taki koleś','wt');
Query OK, 2 rows affected (0.17 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Studenci;
+------------+---------+------------+------------+-------+
| NrStudenta | Imie    | Nazwisko   | Uwagi      | Grupa |
+------------+---------+------------+------------+-------+
|          1 | Alicja  | Zielińska  |            | pn    |
|          2 | Bogdan  | Żółkiewski |            | pn    |
|          3 | Czesław | Ygrekowski | taki koleś | wt    |
+------------+---------+------------+------------+-------+
3 rows in set (0.00 sec)
```

Dodajmy jeszcze więcej danych

```
mysql> INSERT INTO Studenci (Imie,Nazwisko,Grupa) VALUES
    -> ('Dominika','Xena','wt'),
    -> ('Edmund','Woźniak','wf'),
    -> ('Janko','Muzykant','nd');
Query OK, 3 rows affected (0.17 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Studenci;
+------------+----------+------------+------------+-------+
| NrStudenta | Imie     | Nazwisko   | Uwagi      | Grupa |
+------------+----------+------------+------------+-------+
|          1 | Alicja   | Zielińska  |            | pn    |
|          2 | Bogdan   | Żółkiewski |            | pn    |
|          3 | Czesław  | Ygrekowski | taki koleś | wt    |
|          4 | Dominika | Xena       | NULL       | wt    |
|          5 | Edmund   | Woźniak    | NULL       | wf    |
|          6 | Janko    | Muzykant   | NULL       | nd    |
+------------+----------+------------+------------+-------+
6 rows in set (0.04 sec)
```

Zwracam uwagę na różnicę pomiędzy pustymi łańcuchami a `NULL`.

Zapytanie UPDATE

```
mysql> UPDATE Studenci
    -> SET Uwagi=NULL
    -> WHERE NrStudenta=3;
Query OK, 1 row affected (0.19 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE Studenci
    -> SET Grupa='wt' WHERE Grupa='wf';
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Studenci;
+------------+----------+------------+-------+-------+
| NrStudenta | Imie     | Nazwisko   | Uwagi | Grupa |
+------------+----------+------------+-------+-------+
|          1 | Alicja   | Zielińska  |       | pn    |
|          2 | Bogdan   | Żółkiewski |       | pn    |
|          3 | Czesław  | Ygrekowski | NULL  | wt    |
|          4 | Dominika | Xena       | NULL  | wt    |
|          5 | Edmund   | Woźniak    | NULL  | wt    |
|          6 | Janko    | Muzykant   | NULL  | nd    |
+------------+----------+------------+-------+-------+
6 rows in set (0.00 sec)
```

Zapytanie `DELETE`

```
mysql> DELETE FROM Studenci
    -> WHERE Imie='Janko';
Query OK, 1 row affected (0.15 sec)
```

Dodajmy coś jeszcze do tabeli.

```
mysql> INSERT INTO Studenci (Imie, Nazwisko, Grupa) VALUES
    -> ('Feliks','Urban','pn'),('Feliks','Urbański','pn'),
    -> ('Feliks','Urbańczk','wt');
Query OK, 3 rows affected (0.06 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM Studenci;
+------------+----------+------------+-------+-------+
| NrStudenta | Imie     | Nazwisko   | Uwagi | Grupa |
+------------+----------+------------+-------+-------+
|          1 | Alicja   | Zielińska  |       | pn    |
|          2 | Bogdan   | Żółkiewski |       | pn    |
|          3 | Czesław  | Ygrekowski | NULL  | wt    |
|          4 | Dominika | Xena       | NULL  | wt    |
|          5 | Edmund   | Woźniak    | NULL  | wt    |
|          7 | Feliks   | Urban      | NULL  | pn    |
|          8 | Feliks   | Urbański   | NULL  | pn    |
|          9 | Feliks   | Urbańczk   | NULL  | wt    |
+------------+----------+------------+-------+-------+
8 rows in set (0.00 sec)
```

Brakuje numeru '6' — `AUTOINCREMENT` "zapamiętał", że on tam był.

`AUTOINCREMENT` zawsze startuje od *high water mark*. Problem *high water mark* jest jeszcze ważniejszy przy przydzielaniu (i późniejszym *nie*zwalnianiu) bloków pamięci dyskowej.

Jawnie wyspecyfikowana *lista_select*

```
mysql> SELECT Imie, Nazwisko FROM Studenci;
+----------+------------+
| Imie     | Nazwisko   |
+----------+------------+
| Alicja   | Zielińska  |
| Bogdan   | Żółkiewski |
| Czesław  | Ygrekowski |
| Dominika | Xena       |
| Edmund   | Woźniak    |
| Feliks   | Urban      |
| Feliks   | Urbański   |
| Feliks   | Urbańczk   |
+----------+------------+
8 rows in set (0.00 sec)
```

Gwiazdka w zapytaniu `SELECT` (`SELECT * FROM ...`) oznacza “wszystkie kolumny”.

Problem duplikatów

```
mysql> SELECT Imie FROM Studenci;
+----------+
| Imie     |
+----------+
| Alicja   |
| Bogdan   |
| Czesław  |
| Dominika |
| Edmund   |
| Feliks   |
| Feliks   |
| Feliks   |
+----------+
8 rows in set (0.00 sec)
```

```
mysql> SELECT DISTINCT Imie FROM Studenci;
+----------+
| Imie     |
+----------+
| Alicja   |
| Bogdan   |
| Czesław  |
| Dominika |
| Edmund   |
| Feliks   |
+----------+
6 rows in set (0.09 sec)
```

Operator `LIKE`

```
mysql> SELECT Imie, Nazwisko
    -> FROM Studenci
    -> WHERE Nazwisko LIKE 'U%';
+--------+----------+
| Imie   | Nazwisko |
+--------+----------+
| Feliks | Urban    |
| Feliks | Urbański |
| Feliks | Urbańczk |
+--------+----------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT Imie, Nazwisko
    -> FROM Studenci
    -> WHERE Nazwisko LIKE '%i%';
+---------+------------+
| Imie    | Nazwisko   |
+---------+------------+
| Alicja  | Zielińska  |
| Bogdan  | Żółkiewski |
| Czesław | Ygrekowski |
| Edmund  | Woźniak    |
| Feliks  | Urbański   |
+---------+------------+
5 rows in set (0.00 sec)
```

Wykorzystanie `ORDER BY`

```
mysql> SELECT Imie, Nazwisko FROM Studenci
    -> WHERE Grupa='wt'
    -> ORDER BY Nazwisko ASC;
+----------+------------+
| Imie     | Nazwisko   |
+----------+------------+
| Feliks   | Urbańczk   |
| Edmund   | Woźniak    |
| Dominika | Xena       |
| Czesław  | Ygrekowski |
+----------+------------+
4 rows in set (0.09 sec)
```

Tworzenie tabeli wraz z kopiowaniem danych

```
mysql> CREATE TABLE Studenci2
    -> (NrStudenta INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Imie VARCHAR(20) NOT NULL, Nazwisko VARCHAR(20) NOT NULL,
    -> Uwagi VARCHAR(30), Grupa CHAR(2))
    -> -- TU się kończy definicja tabeli!
    -> SELECT * FROM Studenci;
Query OK, 8 rows affected (0.50 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> DESCRIBE Studenci2;
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| NrStudenta | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| Imie       | varchar(20)      | NO   |     | NULL    |                |
| Nazwisko   | varchar(20)      | NO   |     | NULL    |                |
| Uwagi      | varchar(30)      | YES  |     | NULL    |                |
| Grupa      | char(2)          | YES  |     | NULL    |                |
+------------+------------------+------+-----+---------+----------------+
5 rows in set (0.13 sec)
```

```
mysql> INSERT INTo Studenci2 (NrStudenta,Imie,Nazwisko,Grupa) VALUES
    -> (74,'Tadeusz','Kowalski','xy');
Query OK, 1 row affected (0.07 sec)

mysql> SELECT * FROM Studenci2;
+------------+----------+------------+-------+-------+
| NrStudenta | Imie     | Nazwisko   | Uwagi | Grupa |
+------------+----------+------------+-------+-------+
|          1 | Alicja   | Zielińska  |       | pn    |
|          2 | Bogdan   | Żółkiewski |       | pn    |
|          3 | Czesław  | Ygrekowski | NULL  | wt    |
|          4 | Dominika | Xena       | NULL  | wt    |
|          5 | Edmund   | Woźniak    | NULL  | wt    |
|          7 | Feliks   | Urban      | NULL  | pn    |
|          8 | Feliks   | Urbański   | NULL  | pn    |
|          9 | Feliks   | Urbańczk   | NULL  | wt    |
|         74 | Tadeusz  | Kowalski   | NULL  | xy    |
+------------+----------+------------+-------+-------+
9 rows in set (0.00 sec)
```

Aliasy

W zapytaniu `SELECT` możemy nadawać kolumnom inne nazwy niż te, które występują w definicji tabeli. Noszą one nazwę *aliasów*.

```
mysql> SELECT NrStudenta AS ToJestNumer, Imie FROM Studenci2;
+-------------+----------+
| ToJestNumer | Imie     |
+-------------+----------+
|           1 | Alicja   |
|           2 | Bogdan   |
|           3 | Czesław  |
|           4 | Dominika |
|           5 | Edmund   |
|           7 | Feliks   |
|           8 | Feliks   |
|           9 | Feliks   |
|          74 | Tadeusz  |
+-------------+----------+
9 rows in set (0.00 sec)
```

Niekiedy wygodnie jest użyć aliasu w dalszej części zapytania, ale nie można go użyć w kaluzuli `WHERE`.

```
mysql> SELECT NrStudenta AS ToJestNumer, Imie FROM Studenci2
    -> WHERE ToJestNumer > 10;
ERROR 1054 (42S22): Unknown column 'ToJestNumer' in 'where clause'
```

Można natomiast użyć aliasu w klauzuli `HAVING`:

```
mysql> SELECT NrStudenta AS ToJestNumer, Imie FROM Studenci2
    -> HAVING ToJestNumer > 10;
+-------------+---------+
| ToJestNumer | Imie    |
+-------------+---------+
|          74 | Tadeusz |
+-------------+---------+
1 row in set (0.00 sec)
```

Jest to szczególnie przydatne, gdy w warunku występuje wartość obliczana na podstawie więcej niż jednej kolumny:

```
mysql> SELECT CONCAT(Imie,' ',Nazwisko) AS ImieNazwisko FROM Studenci2
    -> HAVING ImieNazwisko LIKE '%s%z%';
+------------------+
| ImieNazwisko     |
+------------------+
| Feliks Urbańczk  |
| Tadeusz Kowalski |
+------------------+
2 rows in set (0.00 sec)
```

Ale to był dopiero początek