BAZY DANYCH
w Biologii i Medycynie

cz. 2 – Język SQL

Adam Piórkowski

pioro@agh.edu.pl

http://home.agh.edu.pl/~pioro/dyd/BDwBiM/

Kraków, 2019-

SQL

Język SQL jest uniwersalnym, standaryzowanym językiem do komunikacji między klientem a bazą danych. Wśród wielu jego dialektów najważniejsze są standardy, w tym ANSI SQL i SQL2 (1992).

SQL nie jest językiem proceduralnym, tylko językiem deklaratywnym, opisowym. Jego użycie polega na definiowaniu poszczególnych komend. Komendy te mogą być uruchamiane w sposób interaktywny (w komunikacji terminalowej między użytkownikiem/administratorem a bazą) lub osadzony (jako zaszyta część aplikacji).

SQL – [skrót ?] - SQL nie jest strukturalny, to nie tylko zapytania

Geneza: SEQUEL (Structured English Query Language), IBM, '70

Składnia SQL

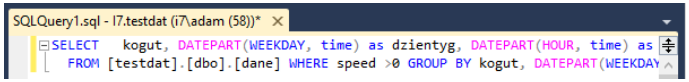
- słowa kluczowe – zbiór ponad 200 słów zarezerwowanych na komendy i operatory; **dla zachowania przejrzystości warto je zapisywać dużymi literami**
- identyfikatory – do rozróżniania utworzonych przez użytkownika obiektów w bazie danych,
- operatory:
  - arytmetyczne,
  - logiczne,
  - porównawcze,
  - tekstowe,
- literały – liczby i łańcuchy znaków, zawarte między apostrofami (lub cudzysłowami),
- znaki porządkowe – { , ` ; ( ) }.

Fraza – zaczyna się od słowa kluczowego komendy, zawiera odpowiedni zestaw słów kluczowych, identyfikatorów, operatorów, literałów i znaków porządkowych.

Zapytanie - składa się z jednej lub kilku fraz (zagnieżdżenie), kończy się znakiem porządkowym (średnik!).

```
select kogut, datepart(weekday, time) as
dzientyg, datepart(hour, time) as godzina,
avg(cast (speed as decimal(5,2))), count(*)
from karetkigps where speed > 0 group by
kogut, datepart(weekday, time), datepart(hour,
time) order by kogut, datepart(weekday, time),
datepart(hour, time);
```

```
SELECT kogut, DATEPART(WEEKDAY, time) as
dzientyg, DATEPART(HOUR, time) as godzina,
AVG(CAST (speed AS DECIMAL(5,2))), COUNT(*)
FROM karetkiGPS WHERE speed > 0 GROUP BY
kogut, DATEPART(WEEKDAY, time), DATEPART(HOUR,
time) ORDER BY kogut, DATEPART(WEEKDAY, time),
DATEPART(HOUR, time);
```



Składnia SQL

- **DDL –** Data Definition Language
  np. `CREATE` (tabele, widoki, itp.), `ALTER` (zmiana), `DROP` (usuwanie)
- **DQL –** Data Query Language
  `SELECT` …
- **DML –** Data Manipulation Language
  `INSERT, UPDATE, DELETE`
  czasem też `SELECT INTO`
- **DCL –** Data Control Language
  `GRANT, REVOKE`

TYPY DANYCH W SQL

Łańcuchy tekstowe

- `CHARACTER` (*długość*) – `CHAR` – łańcuch znaków o długości *długość*, wprowadzone krótsze łańcuchy są dopełniane spacjami.
- `VARCHAR` (*długość*)- `CHARACTER VARYING / CHAR VARYING` – łańcuch znaków o max. długości *długość*, krótsze łańcuchy nie są dopełniane spacjami.
- `NVARCHAR / NATIONAL CHARACTER / NATIONAL CHARACTER VARYING` – ciągi tekstowe w standardzie UNICODE

Łańcuchy bitowe

- `BIT` (*długość*) – łańcuch znaków o długości *długość*, wprowadzone krótsze łańcuchy są dopełniane spacjami.
- `BIT VARYING` (*długość*) – łańcuch znaków o maksymalnej długości *długość*.

TYPY DANYCH W SQL

Liczby dokładne

- NUMERIC – liczba dziesiętna
- DECIMAL - DEC – liczba dziesiętna
- INTEGER - INT – liczba całkowita, zakres ustalony przez system b.d.
- SMALLINT – liczba całkowita o mniejszym zakresie.

Liczby zmiennoprzecinkowe

- FLOAT (*rozdzielczość*) – liczba zmiennoprzecinkowa o rozdzielczości ustalonej przez użytkownika
- REAL (*długość*) – liczba zmiennoprzecinkowa o rozdzielczości ustalonej przez system b.d.
- DOUBLE PRECISION (*długość*) – liczba zmiennoprzecinkowa o największej precyzji.

TYPY DANYCH W SQL

Czas i data

- DATE – data zdefiniowana przez trzy liczby całkowite – rok (YEAR), miesiąc (MONTH) i dzień (DAY), YYYY-MM-DD.
- TIME (rozdzielczość) – czas zdefiniowany przez dwie liczby całkowite – godzina (HOUR), minuty (MINUTE) oraz liczbę dziesiętną sekund (SECOND), HH:MM:SS.SSSS (rozdzielczość)
- DATETIME – tylko w MS SQL

NULL

- NULL – wskazanie nieistnienia danej, nieporównywalne, wiele NULL-i może zostać potraktowane jak duplikat, są grupowane, ich obecność w wyrażeniach rzutuje na wynik wyrażenia równy NULL.

BLOB

- BLOB – Binary Large Object – dane binarne, np. zdjęcia. Nie podlegają interpretacji

Typy danych

TYP	[B]	ZAKRES	OPIS
INTEGER, INT, BIGINT, SMALLINT, TINYINT	4 8,2,1	-2^{31} do 2^{31} -1 -2,147,483,648 - 2,147,483,647	Liczby całkowite
DECIMAL (precyzja, skala) NUMERIC*	~ precyzji		Liczby dziesiętne, np. waluta: 100.15
FLOAT, REAL	Precyzja 4	-1.79E+308 :-2.23E-308, 2.23E-308 to 1.79E+308 - 3.40E + 38 to -1.18E – 38 1.18E - 38 to 3.40E + 38.	Liczby zmiennoprzecinkowe

NUMERIC ma stałą precyzję, DECIMAL – może mieć większą, niż potrzebuje projektant (związane z wydajnością szbd, w części szbd te typy są równoważne

Typy danych

TYP	[B]	ZAKRES	OPIS
CHAR CHAR(20)	1 20	Jeden znak ASCII Tablica 20 znaków ASCII	Jeden znak Dokładnie 20 znaków, dopełniane spacjami
VARCHAR(_X_)	1B - 2GB	Tablica znaków ASCII	Dowolna liczba znaków, ograniczona przez _X_
NVARCHAR(_X_)	2B - 2GB	Tablica znaków UNICODE	Można składować tekst w znakach narodowych
DATE, TIME, DATETIME	10B,..	Np. tekstowo YYYY-MM-DD	czas
BLOB CLOB	Np. do 2GB	Blok binarny Blok tekstowy	Np. na zdjęcia Np. na strony

Rzutowanie

Konwersje niejawne – dozwolone przez system bazy danych.

W przypadku wyrażeń składających się z różnych typów danych, jeżeli możliwa jest konwersja niejawna, to zostanie ona przeprowadzona do najbardziej złożonego typu z użytych typów danych.

Konwersje jawne – konwersje wymuszone przez użytkownika przy pomocy funkcji CAST(). Można dokonywać zmiany liczb (np. dziesiętnych na zmiennoprzecinkowe, zmiennoprzecinkowych na całkowite z utratą części ułamkowej). Podczas operacji zmiany typu danych może zaistnieć błąd – np. nieprawidłowa zawartość tekstu czy przekroczenie zakresu.

Składnia: `CAST ( dana_źródłowa AS typ_wyjściowy )`

Przykład: [kolumna: `speed INT`]

```
SELECT AVG(speed) FROM TrasyKaretek;

SELECT AVG(CAST (speed AS DECIMAL(5,2))) FROM ..
```

Operatory

- **Operatory arytmetyczne**

  W SQL dostępne są proste operatory arytmetyczne {+, -, +, -, *, /},

- **Operatory logiczne**

  W SQL dostępne są operatory logiczne NOT , AND, OR.

- **Operacje na łańcuchach tekstowych**

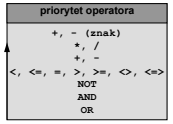
  W SQL operacji na tekstach można dokonać poprzez użycie:

  - operatora złączenia łańcuchów tekstowych ||, (w MySQL jest to odpowiednik OR; w MS Access operator +)

  - zbioru funkcji do operacji na tekstach: { CONCAT(), SUBSTRING(), UPPER(), LOWER(), TRIM(), CHARACTER_LENGTH(), POSITION(), …}

- **Kolejność wykonywania działań**

  Kolejność wykonywania operacji jest wyznaczana w oparciu o priorytet operatora. W przypadku dwóch takich samych operatorów kolejność jest wyznaczana w oparciu o wiązanie (od lewej do prawej). Zmianę kolejności wykonywania operacji można wykonać przy pomocy nawiasów ().



SQL – tworzenie schematów

Polecenie CREATE TABLE służy do tworzenia tabel. Jego składnia jest następująca:

```
CREATE TABLE nazwatabeli ( kol1 typdanych1 [atrybuty1],
  kol2 typdanych1 [atrybuty2] ...)
```

Wśród atrybutów mogą się znaleźć:

- `PRIMARY KEY` – atrybut wchodzi w skład klucza podstawowego,
- `FOREIGN KEY ... REFERENCES` – atrybut wchodzi w skład klucza obcego,
- `NOT NULL` – krotki nie mogą w danej kolumnie zawierać wartości NULL
- `UNIQUE` – w danej kolumnie dozwolone są tylko unikalne wartości,
- `CHECK` – pozwala na wprowadzanie danych spełniających zadane warunki,
- `DEFAULT` – określa wartość domyślną dla danej kolumny,
- `CONSTRAINT` – określa nazwę atrybutu – pomocne w tworzeniu kluczy złożonych, unikalności w obrębie kilku kolumn.

Przykłady:

```
CREATE TABLE Prosta (Wymagany INT NOT NULL, Domyslny INT DEFAULT 7,
   Unikalny INT UNIQUE );
CREATE TABLE ZlozonaUnikalnosc (Rh CHAR(1), Oab CHAR(2), CONSTRAINT Grp
   UNIQUE (Rh, Oab));
CREATE TABLE ZWarunkiem (Ograniczony INT CHECK ( Ograniczony > 100 ) );
```

SQL – tworzenie schematów

Klucz główny prosty

```
CREATE TABLE GlProsty (danap INT PRIMARY KEY);
```

Klucz główny złożony

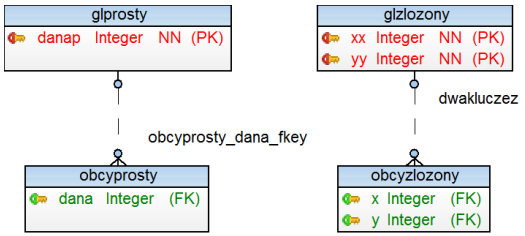
```
CREATE TABLE GlZlozony(xx INT, yy INT,
                       CONSTRAINT klucz_xxyy PRIMARY KEY (xx, yy));
```

Klucz obcy prosty

```
CREATE TABLE ObcyProsty (dana INT REFERENCES GlProsty(danap) );
```

Klucz obcy złożony

```
CREATE TABLE ObcyZlozony ( x INT, y INT, CONSTRAINT dwakluczez
                  FOREIGN KEY (x, y)  REFERENCES GlZlozony (xx,yy) );
```



SQL – tworzenie schematów

Klucz główny prosty

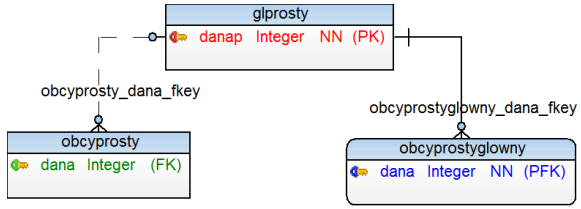
```
CREATE TABLE GlProsty (danap INT PRIMARY KEY);
```

Klucz obcy prosty

```
CREATE TABLE ObcyProsty (dana INT REFERENCES GlProsty(danap) );
```

Klucz obcy prosty jako klucz główny (1:1)

```
CREATE TABLE ObcyProstyGlowny (dana INT PRIMARY KEY REFERENCES
                                          GlProsty(danap) );
```



SQL – tworzenie schematów

Klucz główny złożony

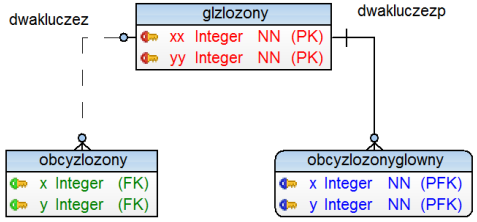
```
CREATE TABLE GlZlozony(xx INT, yy INT,
                       CONSTRAINT klucz_xxyy PRIMARY KEY (xx, yy));
```

Klucz obcy złożony

```
CREATE TABLE ObcyZlozony ( x INT, y INT, CONSTRAINT dwakluczez
                  FOREIGN KEY (x, y)  REFERENCES GlZlozony (xx,yy) );
```

Klucz obcy złożony jako klucz główny

```
CREATE TABLE ObcyZlozonyGlowny ( x INT, y INT,
 CONSTRAINT dwakluczezp FOREIGN KEY (x, y) REFERENCES GlZlozony (xx,yy),
                       CONSTRAINT klucz_pk PRIMARY KEY (x, y) );
```



SQL – tworzenie schematów

Klucz główny prosty

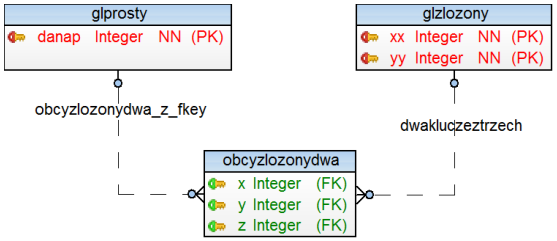
```
CREATE TABLE GlProsty (danap INT PRIMARY KEY);
```

Klucz główny złożony

```
CREATE TABLE GlZlozony(xx INT, yy INT,
                       CONSTRAINT klucz_xxyy PRIMARY KEY (xx, yy));
```

Klucze obce prosty i złożony

```
CREATE TABLE ObcyZlozonyDwa ( x INT, y INT, z INT REFERENCES
        GlProsty(danap), CONSTRAINT dwakluczeztrzech FOREIGN KEY (x, y)
                                    REFERENCES GlZlozony (xx,yy) );
```



Zmiana schematu – ALTER/DROP

Polecenie **`ALTER TABLE`** modyfikuje schemat relacji. Składnia:

```
ALTER TABLE nazwatabeli akcja_modyfikująca
```

gdzie akcja modyfikująca to:

- `ADD [COLUMN] nazwakolumny typdanych`
- `DROP COLUMN nazwakolumny`
- `ADD atrybut`
- `DROP CONSTRAINT atrybut`

Przykład:

```
ALTER TABLE GlProsty ADD dodatkowakolumna INT;
```

`DROP TABLE` usuwa wybraną tabelę z bazy

```
DROP TABLE tabela;
```

`SELECT INTO / CREATE AS`

Przy pomocy konstrukcji SELECT INTO można utworzyć nową tabelę, zawierającą kolumny określone w instrukcji SELECT, wchodzące w skład jednej lub więcej tabel. Nowa tabela od razu jest zapełniana danymi.

Modyfikacja danych (DML)

INSERT

Instrukcja INSERT powoduje wpisanie wiersza danych do bazy. Jej składnia jest następująca:

```
INSERT INTO tabela (kol1, kol2, ...) VALUES (w1, w2, ...);
```

UPDATE

Polecenie UPDATE modyfikuje zawartość wybranych atrybutów krotek, które spełniają zadany warunek. Składnia polecenia UPDATE:

```
UPDATE tabela SET atrybut = wartosc  [WHERE warunek];
```

Przykład:

```
UPDATE Wspolrzedne SET x = x + 2, y = 3 WHERE x > 0 AND y > 0;
```

DELETE

Instrukcja DELETE służy do usuwania całych wierszy z bazy w oparciu o warunek (lub wszystkich, jeśli nie jest podany). Składnia jej jest następująca:

```
DELETE FROM tabela [WHERE warunek];
```

TRUNCATE - usuwa całą zawartość tabeli nie sprawdzając warunków i nie angażując zasobów, np.:

```
TRUNCATE TABLE tabela;
```

Zapytanie SELECT

Zapytanie SELECT służy do wybierania danych z jednej lub wielu tabel, spełniających odpowiednie warunki i sformatowanych w odpowiedni sposób.

Składnia polecenia jest następująca:

SELECT wybrane_kolumny
FROM tabela1 [,tabela2, ...]
[JOIN [<TYP_ZLACZENIA>]]
[WHERE warunek_wybierania]
[GROUP BY kolumn_agrupwania1 [, kolumna_grupowania2, ...]]
[HAVING warunek_filtrowania_grupy]
[ORDER BY kolumna_sortowania1 [, kolumna_sortowania2,...] [ASC DESC]] ;

Zapytanie SELECT

Wyświetlanie pełnej zawartości tabeli:

SELECT * FROM tabela;

Wyświetlanie wybranych kolumn tabeli (projekcja):

SELECT kolumna1, kolumna2 FROM tabela;

Wyświetlanie krotek bez duplikatów:

SELECT DISTINCT * FROM tabela;

Sortowanie wyświetlanych wyników:
do sortowania wyników zapytania służy opcja ORDER BY. Wartości NULL są traktowane specyficznie dla danej implementacji bazy danych.

SELECT * FROM tabela ORDER BY kolumna1 DESC, kolumna2 ASC;

Zapytanie SELECT

Warunek wybierania (opcjonalny) określa się przy pomocy słowa kluczowego WHERE. W warunku tym można stosować operatory arytmetyczne, logiczne, tekstowe oraz specjalne konstrukcje, ułatwiające wyszukiwanie.

SELECT * FROM tabela WHERE kolumna1 > 100;
SELECT * FROM tabela WHERE kolumna1 > 100 OR kolumna2 < 300;
SELECT * FROM tabela WHERE (kolumna1 > 100 OR kolumna2 < 300) AND kolumna 3 <> 0;

Sprawdzanie obecności wartości atrybutu: **IS NULL / IS NOT NULL**

SELECT * FROM tabela WHERE data_obrony IS NULL AND data_odejscia IS NOT NULL;

Sprawdzanie obecności atrybutu w liście wartości:

SELECT * FROM tabela WHERE liczba IN (1,2,3,5,7,11,13,17,19);

Sprawdzanie zakresu wartości atrybutu (przedział domknięty):

SELECT * FROM tabela WHERE liczba BETWEEN 100 AND 200;

Zapytanie SELECT

Tworzenie aliasów kolumn:

```
CREATE TABLE wplaty (id_wplaty INT, kwota DECIMAL(8,2));
INSERT INTO wplaty VALUES(1, 10.0);
INSERT INTO wplaty VALUES(2, 20.0);

SELECT id_wplaty AS numer,
      kwota AS wartosc,
      kwota*0.23 AS „wysokosc podatku”,
      0.23 AS podatek
      FROM wplaty WHERE kwota > 15;

 numer | wartosc | wysokosc podatku | podatek
-------+---------+------------------+---------
     2 |   20.00 |           4.6000 |    0.23
(1 row)
```

Zapytanie SELECT

Operacje na tekstach:

SELECT * WHERE marka BETWEEN ‘Dacia’ AND ‘Ford’;

Operator dopasowywania wzorca:

LIKE:
% - dowolny podciąg znaków, także pusty,
_ - dowolny pojedynczy znak.

SELECT * FROM ksiazki WHERE tytul LIKE ‘%SQL%’;
SELECT * FROM spistresci WHERE rozdzial LIKE ‘_ Postac normalna’;
SELECT * FROM linie WHERE nrautobusu LIKE ‘6__’;

OPERACJE AGREGUJĄCE

Standard SQL określa następujące funkcje agregujące:

MIN(wyrazenie) – wyznacza minimum danego atrybutu z krotek wg określonego wyrażenia,
MAX(wyrazenie) – wyznacza maksimum danego atrybutu z krotek wg określonego wyrażenia,
SUM(wyrazenie) – wyznacza sumę danego atrybutu z krotek wg określonego wyrażenia,
AVG(wyrazenie) – wyznacza średnią wartość danego atrybutu z krotek wg określonego wyrażenia,
COUNT(wyrazenie) – wyznacza liczbę niepustych dla danych atrybutów krotek,

`wyrazenie` – jest wyrażeniem (np. arytmetycznym!) opartym o nazwę kolumny (kolumn).

COUNT (*) - dotyczy także wartości NULL (bezwzględnie wszystkich krotek w tabeli)

Zapytania z funkcjami agregującymi zwracają wartości liczbowe (skalar, wektor). Składnia takich zapytań wymaga, aby owe funkcje występowały jako jedyne cele zapytania SELECT (nie mogą być wyświetlane wraz z atrybutami relacji, chyba, ze z grupowanymi).

```
postgres=# SELECT AVG(liczba) FROM rozneliczby;
        avg
--------------------
 22230.000000000000
(1 row)
```

OPERACJE AGREGUJĄCE

```
postgres =#  SELECT AVG(liczba), MIN(liczba) FROM rozneliczby;
        avg         | min
--------------------+-----
 22230.000000000000 |  50
(1 row)
```

```
postgres=#  SELECT *, AVG(liczba) FROM rozneliczby;
BLAD:  kolumna "rozneliczby.liczba" musi wystepowac w klauzuli GROUP BY lub
 byc uzyta w funkcji agregujacej at character 8
LINE 1: select *, AVG(liczba) from rozneliczby;
                  ^
```

```
postgres=#  SELECT COUNT(*) from rozneliczby;
 count
-------
     5
(1 row)
```

OPERACJE AGREGUJĄCE

```
test(a INT, b INT)

=> SELECT * FROM test;
 a | b
---+---
 1 | 2
 2 | 4
 6 | 8
 2 |
(4 rows)
```

```
=> SELECT MIN (a) FROM test;
 min
-----
   1
(1 row)
```

```
=> SELECT MIN (2*a) FROM test;
 min
-----
   2
(1 row)
```

```
=> SELECT MIN (2*a), AVG(b)
FROM test;
 min |        avg
-----+--------------------
   2 | 4.6666666666666667
(1 row)
```

```
=> SELECT COUNT (*) FROM test;
 count
-------
     4
(1 row)
```

```
=> SELECT COUNT (a) FROM test;
 count
-------
     4
(1 row)
```

```
=> SELECT COUNT (b) FROM test;
 count
-------
     3
(1 row)
```

```
=> SELECT COUNT (a*b) FROM test;
 count
-------
     3
(1 row)
```

GRUPOWANIE DANYCH

Opcja **GROUP BY** `atrybut` wydziela logiczną grupę krotek o tej samej wartości kolumny `atrybut` i pozwala przeprowadzić na tej grupie operacje agregujące dla innych atrybutów.

- atrybut może być kolumną lub listą kolumn,

- dla każdej wartości kolumny atrybut zwracany jest tylko jeden wiersz,

- jeśli w kolumnie atrybut znajduje się NULL, to w wyniku pojawia się wiersz dla wartości NULL,

- wyświetlane w zapytaniu mogą być tylko te atrybuty, które podlegają grupowaniu albo agregacji (inaczej nie wiadomo, jaka wartość miałaby reprezentować atrybut),

Dodatkowa selekcja grupowania odbywa się poprzez opcję `HAVING`. Polecenie to pozwala określić dodatkowy warunek, który musi spełnić krotka grupy, aby pojawiła się w wyniku zapytania.

Kolejność filtrowania: `FROM / JOIN -> GROUP BY -> HAVING`

GRUPOWANIE DANYCH

```
CREATE TABLE
        pracownicy(id_pr INT, plec CHAR, miasto VARCHAR(20), pensja DECIMAL(10,2));
```

```
INSERT INTO pracownicy VALUES(1, 'M', 'Krakow', 2000.00);
INSERT INTO pracownicy VALUES(2, 'K', 'Krakow', 3000.00);
INSERT INTO pracownicy VALUES(3, 'K', 'Krakow', 2000.00);
INSERT INTO pracownicy VALUES(4, 'K', 'Warszawa', 2800.00);
INSERT INTO pracownicy VALUES(5, 'M', 'Warszawa', 3000.00);
INSERT INTO pracownicy VALUES(6, 'm', 'Warszawa', 3100.00);
```

```
SELECT * FROM pracownicy;
 id_pr | plec |  miasto  | pensja
-------+------+----------+--------
     1 | M    | Krakow   | 2000.00
     2 | K    | Krakow   | 3000.00
     3 | K    | Krakow   | 2000.00
     4 | K    | Warszawa | 2800.00
     5 | M    | Warszawa | 3000.00
     6 | m    | Warszawa | 3100.00
(6 rows)
```

```
SELECT *, AVG(pensja) FROM pracownicy GROUP BY miasto;
BLAD:  kolumna "pracownicy.id_pr" musi wystepowac w klauzuli GROUP BY lub byc uzyta
w funkcji agregujacej at character 8
LINE 1: SELECT *, AVG(pensja) FROM pracownicy GROUP BY miasto;
               ^
```

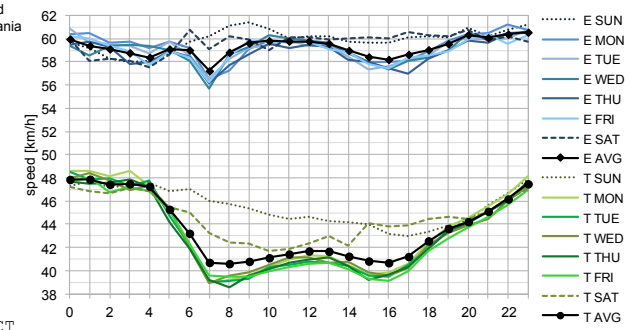
GRUPOWANIE DANYCH

```
SELECT miasto, AVG(pensja) FROM pracownicy GROUP BY miasto;
  miasto  |          avg
----------+-----------------------
 Krakow   | 2333.3333333333333333
 Warszawa | 2966.6666666666666667
(2 rows)

SELECT plec, AVG(pensja), COUNT(*) FROM pracownicy GROUP BY plec;
 plec |          avg          | count
------+-----------------------+-------
 m    | 3100.0000000000000000 |     1
 K    | 2600.0000000000000000 |     3
 M    | 2500.0000000000000000 |     2
(3 rows)

SELECT UPPER(plec), AVG(pensja), COUNT(pensja) FROM pracownicy GROUP BY UPPER(plec);
 upper |          avg          | count
-------+-----------------------+-------
 K     | 2600.0000000000000000 |     3
 M     | 2700.0000000000000000 |     3
(2 rows)

SELECT plec,AVG(pensja),COUNT(*) FROM pracownicy GROUP BY plec HAVING COUNT(plec)>1;
 plec |          avg          | count
------+-----------------------+-------
 K    | 2600.0000000000000000 |     3
 M    | 2500.0000000000000000 |     2
(2 rows)
```

GRUPOWANIE DANYCH

```
SELECT UPPER(plec), miasto, AVG(pensja), COUNT(pensja) FROM pracownicy GROUP BY
UPPER(plec), miasto;
 upper |  miasto  |          avg          | count
-------+----------+-----------------------+-------
 M     | Krakow   | 2000.0000000000000000 |     1
 M     | Warszawa | 3050.0000000000000000 |     2
 K     | Krakow   | 2500.0000000000000000 |     2
 K     | Warszawa | 2800.0000000000000000 |     1
(4 rows)

SELECT UPPER(plec), miasto, AVG(pensja), COUNT(pensja) FROM pracownicy GROUP BY
miasto, UPPER(plec);
 upper |  miasto  |          avg          | count
-------+----------+-----------------------+-------
 M     | Krakow   | 2000.0000000000000000 |     1
 M     | Warszawa | 3050.0000000000000000 |     2
 K     | Warszawa | 2800.0000000000000000 |     1
 K     | Krakow   | 2500.0000000000000000 |     2
(4 rows)

SELECT UPPER(plec), miasto, AVG(pensja), COUNT(pensja) FROM pracownicy GROUP BY
miasto, UPPER(plec) HAVING COUNT(plec) > 1;
 upper |  miasto  |          avg          | count
-------+----------+-----------------------+-------
 M     | Warszawa | 3050.0000000000000000 |     2
 K     | Krakow   | 2500.0000000000000000 |     2
(2 rows)
```

Przykład grupowania




```
SELECT
  kogut,
  DATEPART(WEEKDAY, time) as dzientyg,
  DATEPART(HOUR, time) as godzina,
  AVG(CAST (speed AS DECIMAL(5,2)))
 FROM karetkiGPS WHERE speed > 0
 GROUP BY kogut, DATEPART(WEEKDAY, time), DATEPART(HOUR, time)
 ORDER BY kogut, DATEPART(WEEKDAY, time), DATEPART(HOUR, time);
```

Piórkowski, A. :(Construction of a dynamic arrival time coverage map for emergency medical services. *Open Geosciences*, 2018,*10*(1), 167-173.

Złączenia

```
=> SELECT tabA.atr1, tabA.atr2, tabB.atr1, tabB.atr3 FROM tabA, tabB;
```

albo

```
=>SELECT * FROM tabA, tabB;
```

albo

```
=> SELECT * FROM tabA CROSS JOIN tabB;
```

```
 atr1 | atr2 | atr1 | atr3
------+------+------+------
    1 |   10 |    1 |  100
    1 |   10 |    2 |  200
    1 |   15 |    1 |  100
    1 |   15 |    2 |  200
    2 |   20 |    1 |  100
    2 |   20 |    2 |  200
    3 |   30 |    1 |  100
    3 |   30 |    2 |  200
(8 rows)
```

```
tabA(atr1 INT, atr2 INT);

=> SELECT * FROM tabA;
 atr1 | atr2
------+------
    1 |   10
    1 |   15
    2 |   20
    3 |   30
(4 rows)
```

```
tabB(atr1 INT, atr3 INT);

=> SELECT * FROM tabB;
 atr1 | atr3
------+------
    1 |  100
    2 |  200
(2 rows)
```

Złączenia

```
=> SELECT * FROM tabA, tabB WHERE tabA.atr1 = tabB.atr1;
```

albo

```
=> SELECT * FROM tabA INNER JOIN tabB ON tabA.atr1 = tabB.atr1;
```

```
 atr1 | atr2 | atr1 | atr3
------+------+------+------
    1 |   10 |    1 |  100
    1 |   15 |    1 |  100
    2 |   20 |    2 |  200
(3 rows)
```

```
=> SELECT * FROM tabA NATURAL JOIN tabB;
```

```
 atr1 | atr2 | atr1 | atr3
------+------+------+------
    1 |   10 |    1 |  100
    1 |   15 |    1 |  100
    2 |   20 |    2 |  200
(3 rows)
```

```
tabA(atr1 INT, atr2 INT);

=> SELECT * FROM tabA;
 atr1 | atr2
------+------
    1 |   10
    1 |   15
    2 |   20
    3 |   30
(4 rows)
```

```
tabB(atr1 INT, atr3 INT);

=> SELECT * FROM tabB;
 atr1 | atr3
------+------
    1 |  100
    2 |  200
(2 rows)
```

```
=> SELECT * FROM tabA LEFT OUTER JOIN tabB ON tabA.atr1 = tabB.atr1;
 atr1 | atr2 | atr1 | atr3
------+------+------+------
    1 |   10 |    1 |  100
    1 |   15 |    1 |  100
    2 |   20 |    2 |  200
    3 |   30 |      |
(4 rows)
```

```
=> SELECT * FROM tabA RIGHT OUTER JOIN tabB ON tabA.atr1 = tabB.atr1;
 atr1 | atr2 | atr1 | atr3
------+------+------+------
    1 |   10 |    1 |  100
    1 |   15 |    1 |  100
    2 |   20 |    2 |  200
(3 rows)
```

```
=> SELECT * FROM tabA FULL OUTER JOIN tabB ON tabA.atr1 = tabB.atr1;
 atr1 | atr2 | atr1 | atr3
------+------+------+------
    1 |   10 |    1 |  100
    1 |   15 |    1 |  100
    2 |   20 |    2 |  200
    3 |   30 |      |
(4 rows)
```

```
tabA(atr1 INT, atr2 INT);

=> SELECT * FROM tabA;
 atr1 | atr2
------+------
    1 |   10
    1 |   15
    2 |   20
    3 |   30
(4 rows)
```

```
tabB(atr1 INT, atr3 INT);

=> SELECT * FROM tabB;
 atr1 | atr3
------+------
    1 |  100
    2 |  200
(2 rows)
```

Złączenia

```
-- nazwy atrybutow rozne, celem przejrzystosci - bez kluczy

CREATE TABLE Producenci(id_producenta INT, nazwa_producenta VARCHAR(30));
CREATE TABLE Produkty(id_produktu INT, producent INT, nazwa_produktu VARCHAR(30));

INSERT INTO Producenci VALUES(1, 'Predom Polar');
INSERT INTO Producenci VALUES(2, 'Domgos');
INSERT INTO Produkty VALUES(3, 1, 'Pralka PS 663');
INSERT INTO Produkty VALUES(3, 2, 'Latarka');


bd=> SELECT * FROM Producenci INNER JOIN Produkty ON (id_producenta = producent);
 id_producenta | nazwa_producenta | id_produktu | producent | nazwa_produktu
---------------+------------------+-------------+-----------+----------------
             1 | Predom Polar     |           3 |         1 | Pralka PS 663
             2 | Domgos           |           3 |         2 | Latarka
(2 rows)


bdwbim=> SELECT * FROM Producenci NATURAL JOIN Produkty;
 id_producenta | nazwa_producenta | id_produktu | producent | nazwa_produktu
---------------+------------------+-------------+-----------+----------------
             1 | Predom Polar     |           3 |         1 | Pralka PS 663
             1 | Predom Polar     |           3 |         2 | Latarka
             2 | Domgos           |           3 |         1 | Pralka PS 663
             2 | Domgos           |           3 |         2 | Latarka
(4 rows)
```

Złączenia

```
-- nazwy atrybutow zgodne, celem przejrzystosci - bez kluczy
CREATE TABLE Producenci(id_producenta INT, nazwa_producenta VARCHAR(30));
CREATE TABLE Produkty(id_produktu INT, id_producenta INT, nazwa_produktu
VARCHAR(30));

INSERT INTO Producenci VALUES(1, 'Predom Polar');
INSERT INTO Producenci VALUES(2, 'Domgos');
INSERT INTO Produkty VALUES(3, 1, 'Pralka PS 663');
INSERT INTO Produkty VALUES(3, 2, 'Latarka');

bdwbim=> SELECT * FROM Producenci NATURAL JOIN Produkty;
 id_producenta | nazwa_producenta | id_produktu | nazwa_produktu
---------------+------------------+-------------+----------------
             1 | Predom Polar     |           3 | Pralka PS 663
             2 | Domgos           |           3 | Latarka
(2 rows)

b=> SELECT * FROM Producenci INNER JOIN Produkty ON (id_producenta = id_producenta);
BLAD:  odnosnik kolumny "id_producenta" jest niejednoznaczny at character 50
LINE 1: SELECT * FROM Producenci INNER JOIN Produkty ON (id_producen...

-- dostep kropkowy
bdwbim=> SELECT * FROM Producenci INNER JOIN Produkty
                              ON (Producenci.id_producenta = Produkty.id_producenta);
 id_producenta | nazwa_producenta | id_produktu | id_producenta | nazwa_produktu
---------------+------------------+-------------+---------------+----------------
             1 | Predom Polar     |           3 |             1 | Pralka PS 663
             2 | Domgos           |           3 |             2 | Latarka
(2 rows)
```

Autozłączenie

```
CREATE TABLE Dziesiec(cyfra INT);  -- zlaczenie wielokrotne tej samej tabeli

INSERT INTO Dziesiec VALUES (0);
INSERT INTO Dziesiec VALUES (1);
INSERT INTO Dziesiec VALUES (2);
INSERT INTO Dziesiec VALUES (3);
INSERT INTO Dziesiec VALUES (4);
INSERT INTO Dziesiec VALUES (5);
INSERT INTO Dziesiec VALUES (6);
INSERT INTO Dziesiec VALUES (7);
INSERT INTO Dziesiec VALUES (8);
INSERT INTO Dziesiec VALUES (9);

SELECT dziesiatki.cyfra * 10 + jednostki.cyfra AS liczba
                                    FROM Dziesiec dziesiatki, Dziesiec jednostki;
albo
SELECT dziesiatki.cyfra * 10 + jednostki.cyfra AS "liczba"
                             FROM Dziesiec dziesiatki CROSS JOIN Dziesiec jednostki;

bdwbim=>SELECT dziesiatki.cyfra * 10 + jednostki.cyfra AS liczba FROM Dziesiec
dziesiatki, Dziesiec jednostki;
 liczba
--------
      0
      1
      2
…
     98
     99
(100 rows)
```

Działania na zbiorach

SQL umożliwia działania proste na parach tabel. Ciągi atrybutów muszą odpowiadać sobie pod względem typów.

Suma:

SELECT atr1,...,atrN FROM tabA **UNION [ALL]** SELECT atr1,...,atrN FROM tabB;

Różnica:

SELECT atr1,...,atrN FROM tabA **EXCEPT** SELECT atr1,...,atrN FROM tabB;

Iloczyn:

SELECT atr1,...,atrN FROM tabA **INTERSECT** SELECT atr1,...,atrN FROM tabB;

Zagnieżdżenia

W języku SQL jest możliwe zagnieżdżanie zapytań – umieszczanie zapytań (podzapytań) wewnątrz innych zapytań.

```
SELECT cena FROM tabela WHERE cena > (SELECT AVG(cena) FROM tabela) ;
```

Cechy podzapytania:
- musi być ujęte w nawiasy i nie może być zakończone średnikiem,
- przeważnie występuje we frazie WHERE,
- może być też umieszczone we frazach: SELECT, FROM, HAVING,
- może zawierać odwołania do kolumn wymienionych w zapytaniu zewnętrznym.

```
SELECT peselpracownika FROM zatrudnienie WHERE pensja <
   (SELECT dochody FROM pracownicy WHERE pesel = peselpracownika);
```

tutaj: zatrudnienie (peselpracownika, pensja) , pracownicy (pesel, dochody)

zapytanie zewnętrzne nie może zawierać kolumn tabel wymienionych jedynie w podzapytaniach

```
SELECT peselpracownika FROM zatrudnienie
    WHERE peselpracownika = pesel  AND  pensja <
      (SELECT dochody FROM pracownicy WHERE pesel = peselpracownika);
```

Maksymalna liczba poziomów zagnieżdżeń jest charakterystyczna dla danego systemu baz danych.

Zagnieżdżenia proste i skorelowane

Zagnieżdżenie proste (niezależne) – zapytanie wewnętrzne jest wykonywane tylko raz w czasie wykonywania zapytania zewnętrznego.

Zagnieżdżenie skorelowane – ma miejsce, gdy zapytanie zewnętrzne przekazuje dane do zapytania wewnętrznego – wówczas zapytanie wewnętrzne jest wykonywane dla każdej krotki zapytania zewnętrznego.

```
pracownicy (pesel INT, dochody INT)

 pesel | dochody
-------+---------
     1 |    2000
     2 |    3000
     3 |    4000
     4 |    5000
```

```
zatrudnienie (regonfirmy INT, peselpracownika INT, pensja INT)

regonfirmy | peselpracownika | pensja
-----------+-----------------+--------
       111 |               4 |   5000
       112 |               2 |   3000
       113 |               3 |   3000
       113 |               1 |   2000
```

```
=> SELECT regonfirmy FROM zatrudnienie
     WHERE peselpracownika IN
       (SELECT pesel FROM pracownicy
             WHERE dochody > 3000);

 regonfirmy
------------
        111
        113
```

```
=> SELECT peselpracownika FROM
     zatrudnienie WHERE pensja <
      (SELECT dochody FROM pracownicy
           WHERE pesel = peselpracownika);

 peselpracownika
-----------------
               3
```

Zapytania zagnieżdżone a złączenia

Dla części zapytań zagnieżdżonych można skonstruować równoważne złączenia tabel:

```
SELECT DISTINCT * FROM TabA WHERE kolA1 IN (SELECT kolB1 FROM TabB);

SELECT DISTINCT TabA.* FROM  TabA INNER JOIN TabB ON TabA.kolA1 = TabB.kolB1;
```

```
SELECT * FROM TabA WHERE kolA1 NOT IN (SELECT kolB1 FROM TabB);

SELECT kolA1, kolA2 FROM TabA LEFT OUTER JOIN TabB
                    ON TabA.kolA1 = TabB.kolB1 WHERE TabB.kolB1 IS NULL;
```

Widoki

Język SQL przewiduje tworzenie wirtualnych tabel, opartych o zapytanie SELECT. Tabele te zwane są widokami i są przechowywane w bazie jako definicje zapytań. Ich zbiorem atrybutów są wyszczególnione kolumny z tabeli składowej / tabel składowych, użyte w danym zapytaniu. Widoki posiadają dane – są to krotki wybrane przez instrukcję SELECT. Dane te nie są przechowywane na stałe w bazie. Zestaw krotek jest opracowywany za każdym razem podczas przetwarzania aktualnego zapytania.

Zalety / cele stosowania widoków:
- uproszczony dostęp do danych – łączenie wielu tabel w jedną,
- aktualny stan danych – nie trzeba dokonywać kopiowania danych miedzy tabelami – każde użycie widoku zwraca aktualne dane
- ograniczenie dostępu – możliwość ukrycia wybranych atrybutów przed innymi użytkownikami bez redundancji i problemów aktualizacji,
- możliwość zachowania pewnych funkcjonalności po zmianie logicznej reprezentacji danych,
- widoki mogą enkapsulować skomplikowane zapytania / podzapytania, w szczególności, gdy są używane wielokrotnie,
- widoki mogą być zagnieżdżane.

Składnia tworzenia widoku:

```
CREATE VIEW widok [ (kolumna1, kolumna2, ..) ] AS <Zapytanie_SELECT>;
```

Widoki

```
TabA( t1 INT, t2 INT, t3 INT);

=> SELECT * FROM TabA;
 t1 | t2 | t3
----+----+----
  1 |  2 |  3
  2 |  4 |  6
(2 rows)

=> CREATE VIEW jedensuma AS SELECT t1, t1+t2+t3 AS "kol suma" FROM TabA;

SELECT * FROM jedensuma;
 t1 | kol suma
----+----------
  1 |        6
  2 |       12
(2 rows)
```

Transakcje

```
Tab(a1 INT);


=> BEGIN;
=> UPDATE Tab SET a1 = a1*2;




=> DELETE FROM Tab WHERE a1>3;




=> COMMIT;
```

```
=> SELECT * FROM Tab;
 a1
----
  1
  3
(2 rows)


=> SELECT * FROM Tab;
 a1
----
  2
  6
(2 rows)

=> SELECT * FROM Tab;
 a1
----
  2
(1 row)

=> SELECT * FROM Tab;
 a1
----
  2
(1 row)
```

Transakcje

```
Tab(a1 INT);


=> BEGIN;
=> UPDATE Tab SET a1 = a1*2;




=> DELETE FROM Tab WHERE a1>3;




=> ROLLBACK;
```

```
=> SELECT * FROM Tab;
 a1
----
  1
  3
(2 rows)

=> SELECT * FROM Tab;
 a1
----
  2
  6
(2 rows)

=> SELECT * FROM Tab;
 a1
----
  2
(1 row)

=> SELECT * FROM Tab;
 a1
----
  1
  3
(2 rows)
```

Procedury składowane

Systemy baz danych pozwalają na definiowanie w języku SQL funkcji, które są wywoływane przez klienta a wykonywane (interpretowane) po stronie serwera bazy. Kod samej funkcji może być napisany w zależności od danego systemu w:
- SQL,
- języku kompilowanym do kodu wykonywalnego (np. C/C++),
- języku interpretowanym przez serwer, charakterystycznym dla danej platformy (np. **plpgsql**).

Składnia tworzenia funkcji składowanej w SQL dla bazy PostgreSQL jest następująca:

```
CREATE [ OR REPLACE ] FUNCTION name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    [ RETURNS rettype ]
  { LANGUAGE langname
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

Procedury składowane

```
CREATE FUNCTION kwadratsumy(integer, integer)
  RETURNS integer
    AS 'select $1*$1 + 2*$1*$2 + $2*$2;'
  LANGUAGE SQL;

=>SELECT kwadratsumy(1,2);

 kwadratsumy
-------------
           9
(1 row)


DROP FUNCTION kwadratsumy(integer, integer);
```

Procedury składowane

```
=> CREATE FUNCTION dodatnie_suma()
    RETURNS NUMERIC
     AS
       'UPDATE dane SET a = -a WHERE a <0;
        SELECT AVG (a) FROM dane;'
     LANGUAGE SQL;

⇒SELECT dodatnie_suma();

    dodatnie_suma
--------------------
 2.0000000000000000
(1 row)

=> DROP FUNCTION dodatnie_suma();
```

```
=> CREATE TABLE dane(a INT);

=> SELECT * FROM dane;

 a
----
  1
  2
 -3
(3 rows)
```

Procedury składowane w innych językach (PL/pgSQL, PL/Tcl, PL/Perl) w PostgreSQL

```
CREATE FUNCTION sum_n_product(x int, y int, OUT sum int, OUT prod int)
AS $$
BEGIN
    sum := x + y;
    prod := x * y;
END;
$$ LANGUAGE plpgsql;
```
www.postgresql.org

```
CREATE FUNCTION tcl_max(integer, integer) RETURNS integer AS $$
    if {[argisnull 1]} {
        if {[argisnull 2]} { return_null }
        return $2
    }
    if {[argisnull 2]} { return $1 }
    if {$1 > $2} {return $1}
    return $2
$$ LANGUAGE pltcl;
```
www.postgresql.org

```
CREATE FUNCTION perl_max (integer, integer) RETURNS integer AS $$
    if ($_[0] > $_[1]) { return $_[0]; }
    return $_[1];
$$ LANGUAGE plperl;
```
www.postgresql.org

Wyzwalacze (triggers)

Język SQL pozwala na definiowanie akcji użytkownika podczas wybranych operacji w bazie. Takim narzędziem są wyzwalacze. Są one wywoływanie podczas zmian zawartości tabel instrukcjami INSERT, UPDATE, DELETE. Definicja wyzwalacza w SQL jest następująca:

```
CREATE TRIGGER nazwatr { BEFORE | AFTER } { INSERT OR UPDATE OR DELETE }
  tabela FOR EACH { ROW|STATEMENT } EXECUTE PROCEDURE proctriggera
     ( parametry );
```

```
CREATE TABLE uczniowie(a INT);
CREATE TABLE nowiuczniowie(a INT);

CREATE OR REPLACE FUNCTION funkcjawyzwalacza ()
RETURNS TRIGGER AS  '
   BEGIN
      INSERT INTO nowiuczniowie VALUES (1);
      return NEW;
   END;
' LANGUAGE  'plpgsql' ;

INSERT INTO uczniowie VALUES(1);
(1)
CREATE TRIGGER wyzwalacz AFTER INSERT ON uczniowie
   FOR EACH ROW EXECUTE PROCEDURE funkcjawyzwalacza ();

INSERT INTO uczniowie VALUES(2);
(2)
```

```
(1)

 SELECT * FROM
        nowiuczniowie;
 a
---
(0 rows)


(2)

SELECT * FROM
        nowiuczniowie;
 a
---
 1
(1 row)
```

SQL – praktyka … psql

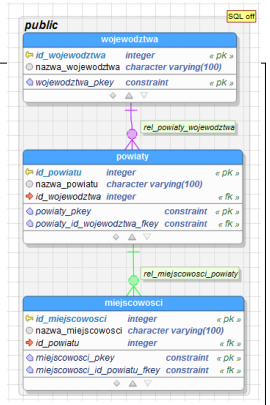
```
baza=>
baza=> SELECT * FROM korelacje
baza-> WHERE ( wymiar > 2 AND
baza(>
baza(> normalizacja IS NULL )
baza->
baza->
baza-> ;
 wymiar | normalizacja | korelacja
--------+--------------+-----------
(0 rows)

baza=>
```

SQL – praktyka …

```
CREATE TABLE Wojewodztwa(id_wojewodztwa INT PRIMARY KEY,
                         nazwa_wojewodztwa VARCHAR(100));
CREATE TABLE Powiaty(id_powiatu INT PRIMARY KEY,
      nazwa_powiatu VARCHAR(100), id_wojewodztwa INT REFERENCES
                                  Wojewodztwa(id_wojewodztwa) );
CREATE TABLE Miejscowosci(id_miejscowosci INT PRIMARY KEY,
     nazwa_miejscowosci VARCHAR(100), id_powiatu INT REFERENCES
                                      Powiaty(id_powiatu));

INSERT INTO Wojewodztwa(id_wojewodztwa, nazwa_wojewodztwa)
                                      VALUES(1,'Malopolskie');
INSERT INTO Wojewodztwa VALUES(2,'Opolskie');
INSERT INTO Wojewodztwa VALUES(3,'Podkarpackie'), (4, 'Slaskie');

INSERT INTO Powiaty VALUES(1, 'Krakowski', 1);
INSERT INTO Powiaty VALUES(2, 'Wielicki', 1);
INSERT INTO Powiaty VALUES(3, 'Nyski', 2);

INSERT INTO Miejscowosci VALUES(1, 'Skawina', 1);
INSERT INTO Miejscowosci VALUES(200, 'Wieliczka', 2);
INSERT INTO Miejscowosci VALUES(300, 'Nysa', 3);
```

SQL – praktyka …

```
bdwbim=> SELECT * FROM Wojewodztwa;
 id_wojewodztwa | nazwa_wojewodztwa
----------------+-------------------
              1 | Malopolskie
              2 | Opolskie
              3 | Podkarpackie
              4 | Slaskie
(4 rows)

bdwbim=> SELECT * FROM Powiaty;
 id_powiatu | nazwa_powiatu | id_wojewodztwa
------------+---------------+----------------
          1 | Krakowski     |              1
          2 | Wielicki      |              1
          3 | Nyski         |              2
(3 rows)

bdwbim=> SELECT * FROM Miejscowosci;
 id_miejscowosci | nazwa_miejscowosci | id_powiatu
-----------------+--------------------+------------
               1 | Skawina            |          1
             200 | Wieliczka          |          2
             300 | Nysa               |          3
(3 rows)
```




```
bdwbim=> SELECT * FROM Wojewodztwa INNER JOIN Powiaty ON Wojewodztwa.id_wojewodztwa
= Powiaty.id_wojewodztwa;
 id_wojewodztwa | nazwa_wojewodztwa | id_powiatu | nazwa_powiatu | id_wojewodztwa
----------------+-------------------+------------+---------------+----------------
              1 | Malopolskie       |          1 | Krakowski     |              1
              1 | Malopolskie       |          2 | Wielicki      |              1
              2 | Opolskie          |          3 | Nyski         |              2
(3 rows)

bdwbim=> SELECT * FROM Wojewodztwa NATURAL JOIN Powiaty;
 id_wojewodztwa | nazwa_wojewodztwa | id_powiatu | nazwa_powiatu
----------------+-------------------+------------+---------------
              1 | Malopolskie       |          1 | Krakowski
              1 | Malopolskie       |          2 | Wielicki
              2 | Opolskie          |          3 | Nyski
(3 rows)

-- ZAPYTANIA ŁĄCZĄCE 3 TABELE (lub więcej)

bdwbim=> SELECT * FROM Wojewodztwa NATURAL JOIN Powiaty NATURAL JOIN Miejscowosci;
 id_powiatu | id_wojewodztwa | nazwa_wojewodztwa | nazwa_powiatu | id_miejscowosci | nazwa_miejscowosci
------------+----------------+-------------------+---------------+-----------------+--------------------
          1 |              1 | Malopolskie       | Krakowski     |               1 | Skawina
          2 |              1 | Malopolskie       | Wielicki      |             200 | Wieliczka
          3 |              2 | Opolskie          | Nyski         |             300 | Nysa
(3 rows)

bdwbim=> SELECT * FROM Wojewodztwa INNER JOIN Powiaty ON Wojewodztwa.id_wojewodztwa =
Powiaty.id_wojewodztwa INNER JOIN Miejscowosci ON Powiaty.id_powiatu = Miejscowosci.id_powiatu;
 id_wojewodztwa | nazwa_wojewodztwa | id_powiatu | nazwa_powiatu | id_wojewodztwa | id_miejscowosci | nazwa_miejscowosci | id_powiatu
----------------+-------------------+------------+---------------+----------------+-----------------+--------------------+------------
              1 | Malopolskie       |          1 | Krakowski     |              1 |               1 | Skawina            |          1
              1 | Malopolskie       |          2 | Wielicki      |              1 |             200 | Wieliczka          |          2
              2 | Opolskie          |          3 | Nyski         |              2 |             300 | Nysa               |          3
(3 rows)
```