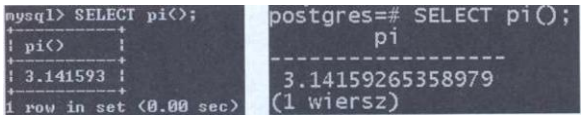
FUNKCJE W BAZACH DANYCH

Funkcje matematyczne
Zadaniem funkcji matematycznych jest umożliwienie wykonania operacji na liczbach, zwłaszcza gdy zachodzi potrzeba wykonania operacji bardziej skomplikowanych, takich jak wypisanie reszty z dzielenia, wartości bezwzględnej, wyciągnięcie pierwiastka czy podniesienie do potęgi.
Działanie funkcji matematycznych możemy zaobserwować na przykładzie MySQl_ oraz PostgreSQL



Funkcja	Typ	Opis	Przykad	Wynik
abs(x)	-	wartość bezwzględna	abs(-17.4)	17,4
cbrt(dp) DOUBLE PRECISION	dp	pierwiastek 3 st	cbrt(27.0)	3
ceil(dp or numeric)	-	część całkowita	ceil(-42.8)	-42
ceiling(dp or numeric)	-	część cakowita	ceiling(-95.3)	-95
degrees(dp)	dp	zamiana rad na st.	degrees(0.5)	28,65
exp(dp or numeric)	-	exponential	exp(1.0)	2,72
floor(dp or numeric)	-	część całkowita	floor(-42.8)	-43
ln(dp or numeric)	-	logarytm naturalny	ln(2.0)	0,69
log(dp or numeric)	-	logarytm dziesiętny	log(100.0)	2
log(b numeric, x numeric)	numeric	log przy podstawie b	log(2.0, 64.0)	6
mod(y, x)	-	reszta z dzielenia	mod(9,4)	1
pi()	dp	liczba PI	pi()	3,14
power(a dp, b dp)	dp	a raised to the power of b	power(9.0, 3.0)	729
power(a numeric, b numeric)	numeric	a raised to the power of b	power(9.0, 3.0)	729
radians(dp)	dp	zamiana st. na rad	radians(45.0)	0,79
random()	dp	liczba losowa z przedziału 0-1	random()	-
round(dp or numeric)	-	zaokrąglenie	round(42.4)	42
round(v numeric, s integer)	numeric	zaokrągenie z podaną precyzją	round(42.4382, 2)	42,44
setseed(dp)	integer	set seed for subsequent random() calls	setseed(0.54823)	1177314959
sign(dp or numeric)	-	znak liczby	sign(-8.4)	-1
sqrt(dp or numeric)	-	pierwiastek	sqrt(2.0)	1,41
trunc(dp or numeric)	-	liczba całkowita	trunc(42.8)	42
trunc(v numeric, s integer)	numeric	obcięcie z dokładnością do..	trunc(42.4382, 2)	42,43

Funkcje tekstowe

Podobnie jak funkcje matematyczne wykonują określone operacje na liczbach, tak funkcje tekstowe używane są do wykonywania operacji na ciągach tekstowych. Przykładem takiej operacji może być usuwanie wielkich liter z tekstu, zmiana wszystkich znaków na wielkie litery itd. Funkcje tekstowe przyjmują pewien parametr wejściowy, przetwarzają go i zwracają wynik. Parametrem wejściowym jest w tym wypadku tekst.
Gdy dysponujemy dużą bazą danych, może zajść potrzeba przetwarzania tekstów. Gdy w tabeli znajduje się kilka tysięcy nazwisk i osoba wprowadzająca dane na początku nie- których nazwisk stawiała spacje (które trzeba teraz usunąć), można zastosować funkcję LTRIM(). Np.: http://www.sqlpedia.pl/funkcje-tekstowe-sql-stringi/

FUNKCJE AGREGUJĄCE(rodzaj funkcji statystycznych) działają na zbiorach rekordów, nazywanych grupami (w przeciwieństwie do funkcji wierszowych, które zawsze działają na jednym rekordzie).

Przed zastosowaniem funkcji agregującej konieczne jest podzielenie rekordów na grupy, tzw. grupowanie. Do jednej grupy należą te rekordy relacji, dla których tzw. wyrażenie grupujące zwraca tą samą wartość. Wyrażeniem grupującym jest najczęściej pojedynczy atrybut relacji.

Po podziale rekordów na grupy w każdej z grup zostaje zastosowana funkcja agregująca, która wylicza pojedynczą wartość dla grupy. Stąd w wyniku zapytania otrzymujemy tyle rekordów, ile grup zostało utworzonych w wyniku operacji grupowania.

FUNKCJE AGREGUJĄCE

AVG - średnia (niepustych wartości)
COUNT - Liczba wartości w wyrażeniu
COUNT(*) - Liczba zaznaczonych wierszy
First i Last – pierwszy i ostatni wiersz
MAX - Największa wartość w wyrażeniu
MIN - Najmniejsza wartość w wyrażeniu
SUM - Suma wartości w wyrażeniu numerycznym
STDEV - Odchylenie statystyczne dla wszystkich wartości
STDEVP - Odchylenie statystyczne dla populacji
VAR - Wariancja statyczna dla wszystkich wartości
VARP - Wariacja statyczna dla wszystkich wartości w populacji

GROUP BY
Klauzula GROUP BY używana jest do zestawiania krotek posiadających te same wartości w kolumnach określonych przez klauzule.

Tabela zawody_wedkarskie (struktura)

```
mojabaza=# \d zawody_wedkarskie;
          Tabela "public.zawody_wedkarskie"
        Kolumna        |          Typ          | Modyfikatory
-----------------------+-----------------------+--------------
 id_zawodnika          | integer               | niepusty
 imie                  | character varying(50) |
 nazwisko              | character varying(50) |
 liczba_zlowionych_ryb | integer               |

mojabaza=# _
```

Polecenie SELECT * FROM zawody_wedkarskie (dane czyli zawartość)zwraca wynik:

```
mojabaza=# SELECT * FROM zawody_wedkarskie;
 id_zawodnika |   imie   |   nazwisko   | liczba_zlowionych_ryb
--------------+----------+--------------+-----------------------
            1 | Ireneusz | Kowalski     |                     2
            2 | Mateusz  | Wielgosz     |                     2
            3 | Tadeusz  | Nowak        |                     5
            4 | Marian   | Zapominalski |                     2
            5 | Mirek    | Kowalski     |                     2
            6 | Marcin   | Nowak        |                     2
            7 | Jan      | Nowak        |                     3
            8 | Zbigniew | Kowalski     |                     6
            9 | Jan      | Wonski       |                     3
           10 | Edward   | Siarzewski   |                     6
(10 wierszy)
```

Jeśli zechcemy pogrupować dane, za kryterium przyjmując liczbę złowionych ryb, wówczas użyjemy klauzuli GROUP BY, np.

```
SELECT liczba_zlowionych_ryb,COUNT(id_zawodnika)
FROM zawody_wedkarskie
GROUP BY liczba_zlowionych ryb;
```

(5) SELECT
(1) FROM
(2) WHERE
(3) GROUP BY
(4) HAVING
(6) ORDER BY

```
mojabaza=# SELECT liczba_zlowionych_ryb,COUNT(id_zawodnika)
mojabaza=# FROM zawody_wedkarskie
mojabaza=# GROUP BY liczba_zlowionych_ryb;
 liczba_zlowionych_ryb | count
-----------------------+-------
                     6 |     3
                     2 |     5
                     3 |     2
(3 wiersze)
```

Jeśli zachodzi konieczność, by z wyników zwracanych przez **GROUP BY** wykluczyć pewien rodzaj danych lub upewnić się, że wyświetlony zostanie tylko interesujący nas fragment danych otrzymany za pomocą **GROUP BY**, używamy klauzuli **HAVING**

HAVING (odpowiednik klauzuli WHERE dla funkcji agregujacych)
Dla poprzednio omawianego przykładu zainteresuje nas tylko liczba zawodników, którzy złowili sześć ryb.

```
mojabaza=# SELECT liczba_zlowionych_ryb,COUNT(id_zawodnika)
mojabaza-# FROM zawody_wedkarskie
mojabaza-# GROUP BY liczba_zlowionych_ryb
mojabaza-# HAVING MAX(liczba_zlowionych_ryb) = 6;
 liczba_zlowionych_ryb | count
-----------------------+-------
                     6 |     3
(1 wiersz)
```

Należy pamiętać, że klauzula **HAVING**, oprócz tego że musi być poprzedzona klauzulą **GROUP** musi być typu logicznego .

(5) SELECT
(1) FROM
(2) WHERE
(3) GROUP BY
(4) HAVING
(6) ORDER BY

Funkcje agregujące (przykłady)
Funkcje agregujące zaprezentujemy na przykładzie funkcji **SUM**. Wyobraźmy sobie hipotetyczną sytuację, w której udzielamy kredytu trzem osobom: Piotrkowi, Maćkowi i Justynie.

Kolumna zadłużenie zawiera kwoty, które poszczególne osoby są nam winne. Jeśli chcielibyśmy się dowiedzieć, jaka jest całkowita wartość zadłużenia, możemy użyć funkcji agregującej SUM, która jako parametr przyjmie nazwę atrybutu zadłużenie i zwróci zsumowane wartości 500, 200, 300

```
postgres=# SELECT * FROM kredyt;
 zadłużenie |  imię   | numer
------------+---------+-------
        500 | Piotrek |     1
        200 | Maciek  |     2
        300 | Justyna |     3
(3 rows)

postgres=# SELECT sum(zadłużenie) FROM kredyt;
 sum
------
 1000
(1 row)

postgres=#
```

Podczas obliczania statystyk znacznie bardziej przydatną informacją może okazać się średnia wieku pracowników firmy. W tym celu posłużymy się słowem kluczowym **AVG**

```
mojabaza=# SELECT AVG(wiek)
mojabaza-# AS SREDNIA_WIEKU
mojabaza-# FROM pracownicy_bhp;
    srednia_wieku
----------------------
 27.2000000000000000
(1 wiersz)
```

Aby określić wiek najstarszego i najmłodszego pracownika, posłużymy się funkcjami **MAX** i **MIN**.

```
mojabaza=# SELECT MAX(wiek),MIN(wiek)
mojabaza-# FROM pracownicy_bhp;
 max | min
-----+-----
  33 |  23
(1 wiersz)
```

W dużych tabelach przydatna bywa również funkcja przeliczająca **COUNT**, która może podać liczbę pracowników na podstawie przeliczonej liczby krotek tabeli.

```
mojabaza=# SELECT COUNT(id_pracownika) AS "Liczba pracownikow" FROM pracownicy_bhp;
 Liczba pracownikow
--------------------
                  5
(1 wiersz)
```

Jeśli w tabeli nie występują wartości **NULL**, możemy posłużyć się następującą konstrukcją zapytania:

```
mojabaza=# SELECT COUNT(*) AS "Liczba pracownikow" FROM pracownicy_bhp;
 Liczba pracownikow
--------------------
                  5
(1 wiersz)
```

Funkcje daty i czasu

Analogicznie do poprzednich funkcji, funkcje operacji na datach używane są do manipulowania datami. W bazach danych funkcje daty i czasu znajdują zastosowanie, gdy np. trzeba wyszukać pracownika firmy na podstawie obliczeń czasu (zaległe dni urlopu, liczba dni do emerytury itd.).

Każda baza danych może mieć nieco inny zestaw funkcji daty i czasu, dlatego warto nie tylko zapoznać się z dokumentacją, lecz także śledzić zmiany po wydaniach kolejnej wersji oprogramowania.
Aby poznać aktualną datę w PostgreSQL używamy następującego polecenia:

```
mojabaza=# SELECT current_date;
```

Czas uzyskujemy za pomocą funkcji **current_time**.

```
mojabaza=# SELECT current_time;
     timetz
-----------------
 03:36:33.134+02
(1 wiersz)
```

Funkcje matematyczne pozwalają na dokonywanie obliczeń, gdy jako parametr wejścio-wy podajemy czas lub przedziały czasowe. Przykładem jest funkcja **AGE** (timestamp ' RRRR-MM-DD ' , timestamp ' RRRR-MM-DD 1) odejmująca przedziały czasowe. Funkcje daty czasu potrafią, jako parametry wejściowe, przyjmować dane z kolumn tabel i prowadzić na tych danych obliczenia. Przykładem wprowadzania danych do funkcji jest użycie funkcji **AGE** przedstawione na ilustracji:

```
mojabaza=# SELECT age(timestamp '2012-10-10', timestamp '1994-09-10');
       age
-----------------
 18 years 1 mon
(1 wiersz)
```

Funkcje sterowania przepływem

PostgreSQL, podobnie jak MySQL, zawiera funkcje sterowania przepływem.
Możemy wykorzystywać je w poleceniach SQL. aby optymalizować pytania i precyzować wyniki.
Jedną z funkcji sterowania przepływem jest **CASE**. Praktyczne użycie tego polecenia przedstawiają poniższe ilustracje. Składnia polecenia to

```
CASE wartość
WHEN [warunek] THEN wynik
     [warunek] THEN wynik
...
[ELSE wynik]
END
```

```
mojabaza=# select * from zawodnicy;
 nr |  imie   | wiek
----+---------+------
  1 | Janek   |   12
  2 | Marek   |   15
  3 | Heniek  |   22
  4 | Kornel  |   34
  5 | Wojtek  |   25
  6 | Tadeusz |   43
(6 wierszy)
```

```
mojabaza=# select nr,
mojabaza-# CASE WHEN nr=1 THEN 'jeden
mojabaza-# WHEN nr=2 THEN 'dwa'
mojabaza-# else 'nic'
mojabaza-# END
mojabaza-# FROM ZAWODNICY;
 nr | case
----+-------
  1 | jeden
  2 | dwa
  3 | nic
  4 | nic
  5 | nic
  6 | nic
(6 wierszy)
```

Oprócz **CASE** MySQL obsługuje takie funkcje jak:

Tabela 16.2. Funkcje obsługiwane przez MySQL

IF()	IF (warunek, wynik1, wynik2) – gdy warunek jest spełniony, funkcja zwraca wynik1, w przeciwnym wypadku wynik2.
IFNULL()	IFNULL(warunek, wynik) – funkcja zwraca wynik warunku będącego pierwszym argumentem, gdy wynik jest różny od NULL.
NULLIF()	NULLIF(warunek1, warunek2) – funkcja zwraca NULL, gdy podane argumenty są równe.

```
select  age((select current_date), timestamp '1967-10-11');
```