MINISTERSTWO ŚRODOWISKA

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

TRAPPA

Szkolenie
Język SQL w bazie PostgreSQL

Część praktyczna

Prowadzący

Michał Bednarczyk

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

SQL wprowadzenie

- SQL jest standardowym językiem wykonywania zapytań i manipulowania danymi oraz zarządzania bazą danych
- SQL jest **językiem wysokiego poziomu**
  - Działa dobrze, ponieważ jest wysoce zoptymalizowany
- Istnieje kilka standardów SQL:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
  - Wytwórcy oprogramowania implementują różne z nich

SQL czyli
Structured Query Language

Prawdopodobnie najbardziej udany język przetwarzania równoległego (i wieloprocesorowego)

SQL jest..

- Językiem deklaratywnym (w przeciwieństwie do imperatywnego np. Python'a)
- Językiem definiowania danych (Data Definition Language - DDL)
  - Służy do definiowania schematów relacyjnych
  - Służy do tworzenia, aktualizowania struktury tabel oraz i usuwania tabel
- Językiem manipuacji danych (Data Manipulation Language - DML)
  - Służy do wstawiania, kasowania i modyfikowania wierszy w tabelach (krotek)
  - Służy do wyszukiwania danych (wykonywania zapytań) w jednej lub wielu tabelach

Tabele w SQL

Produkt

PNazwa	Cena	Producent
Aspire	2340	Acer
SP300	3560	Acer
Inspiron	3200	Dell
MultiTouch	12400	Hitachi

Relacja lub tabela jest multizbiorem krotek posiadających atrybuty określone zgodnie ze schematem

Tabele w SQL

Produkt

PNazwa	Cena	Producent
Aspire	2340	Acer
SP300	3560	Acer
Inspiron	3200	Dell
MultiTouch	12400	Hitachi

Multizbiór jest nieuporządkowaną listą (lub też wielowartościowym zbiorem, w którym dopuszcza się powtarzanie wartości)

Lista: [1, 1, 2, 3]
Zbiór: {1, 2, 3}
Multizbiór: {1, 1, 2, 3}

Tabele w SQL

Produkt

PNazwa	Cena	Producent
Aspire	2340	Acer
SP300	3560	Acer
Inspiron	3200	Dell
MultiTouch	12400	Hitachi

Atrybut (lub kolumna) wartość określonego typu, utożsamiana z cechą, obecna jest w każdej krotce relacji

*Atrybuty w standardzie SQL muszą być typu **atomowego**, czyli bez list, zbiorów itp.*

Tabele w SQL

Produkt

PNazwa	Cena	Producent
Aspire	2340	Acer
SP300	3560	Acer
Inspiron	3200	Dell
MultiTouch	12400	Hitachi

Czasem używa się też określenia ***rekord***

Krotka lub wiersz to pojedyncza część tabeli, posiadająca atrybuty określone w schemacie

Schematy tabel

- **Schemat** tabeli określa się poprzez nazwę, atrybuty i ich typy

```
Produkt(Pnazwa: string, Cena: float, Kategoria:
string, Producent: string)
```

Klucz jest atrybutem, którego wartości są unikalne, jego nazwa jest podkreślona

```
Produkt(Pnazwa: string, Cena: float, Kategoria:
string, Producent: string)
```

Klucz - ograniczenie integralnościowe

Klucz jest **minimalnym podzbiorem atrybutów**, jednoznacznie identyfikującym krotki w relacji (wiersze w tabeli)

- Klucz jest bezwarunkowym ograniczeniem według którego krotki są w relacji
  - np. jeżeli dwie krotki mają tę samą wartość klucza, to muszą być tą samą krotką!

```
Studenci(sid:string, nazwisko:string,
srednia: float)
```

1. Którą wartość wybierzemy jako klucz?
2. Czy istnienie klucza jest zawsze zagwarantowane?
3. Czy może istnieć więcej niż jeden klucz?

NULL i NOT NULL

- Aby powiedzieć “nie znam wartości” użyjemy wartości `NULL`

```
Students(sid:string, nazwisko:string, srednia: float)
```

sid	nazwisko	srednia
114	Nowak	3.9
115	Kowalski	NULL

W SQL możemy ograniczyć kolumnę do bycia niepustą (NOT NULL), jak np. nazwisko w powyższym przykładzie

Zapytania jednotabelowe

Zapytanie SQL

- Najprostsza forma

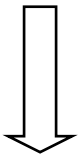
```
SELECT <atrybuty>
FROM   <jedna lub więcej tabel/relacji>
WHERE  <warunki>
```

Przykładowe zapytanie SQL: Selekcja

Selekcja polega na filtrowaniu wierszy według podanego warunku

PNazwa	Cena	Kategoria	Producent
Aspire	2340	Laptop	Acer
SP300	3560	Laptop	Acer
Inspiron	3200	Desktop	Dell
MultiTouch	12400	Tablet	Hitachi

```
SELECT *
FROM   Produkt
WHERE  Kategoria = 'Laptop'
```



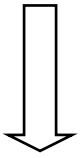
PNazwa	Cena	Kategoria	Producent
Aspire	2340	Laptop	Acer
SP300	3560	Laptop	Acer

Zapytanie SQL: Projekcja

W **projekcji** tabela wyjściowa zawiera krotki, które posiadają podzbiór swoich pierwotnych atrybutów

PNazwa	Cena	Kategoria	Producent
Aspire	2340	Laptop	Acer
SP300	3560	Laptop	Acer
Inspiron	3200	Desktop	Dell
MultiTouch	12400	Tablet	Hitachi

```
SELECT PNazwa, Cena, Producent
FROM   Produkt
WHERE  Kategoria = 'Laptop'
```



PNazwa	Cena	Producent
Aspire	2340	Acer
SP300	3560	Acer

Operatory arytmetyczne w SQL

Operator	Działanie
+	dodawanie
-	odejmowani
*	mnożenie
/	dzielenie
%	modulo

Operatory porównania w SQL

Operator	Działanie
=	równe
>	większe
<	mniejsze
>=	większe lub równe
<=	mniejsze lub równe
<>	różne

Operatory logiczne w SQL

Operator	Działanie (zwracana wartość logiczna)
ALL	TRUE jeżeli wszystkie wartości podzapytania spełniają warunek
AND	TRUE jeżeli wszystkie warunki rozdzielone AND są TRUE
ANY	TRUE jeżeli którakolwiek z wartości podkwerendy spełnia warunek
BETWEEN	TRUE jeżeli operand znajduje się w podanym przedziale porównania
EXISTS	TRUE jeżeli podzapytanie zwraca jeden lub więcej rekordów
IN	TRUE jeżeli operand jest równy jednej z wartości znajdującej się na liście
LIKE	TRUE jeżeli operand pasuje do wzoru (maski)
NOT	Wyświetla rekord jeżeli warunek (warunki) jest niespełniony (NOT TRUE)
OR	TRUE jeżeli którykolwiek z warunków rozdzielonych OR jest TRUE
SOME	TRUE jeżeli którakolwiek z wartości podzapytania spełnia warunek

Operator IS

Jeżeli chcemy wykryć (lub uwzględnić) rekord zawierający wartość NULL użyjemy w zapytaniu operatora IS:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM   Osoba
WHERE  wiek < 25 OR wiek >= 25
    OR wiek IS NULL
```

Wyrażenie LIKE: maska wyszukiwania

```
SELECT *
FROM   Produkt
WHERE  PNazwa LIKE '%laptop%'
```

- s **LIKE** p: maska wyszukiwania wartości teksowych (string)
- p może zawierać znaki specjalne:
  - % = dowolny ciąg znaków
  - _ = dowolny pojedynczy znak

Aliasy atrybutów

Tworząc zapytanie można użyć wyrażenia **AS** aby nadać **własną nazwę** kolumnie wynikowej.

PNazwa	Cena	Kategoria	Producent
Aspire	2340	Laptop	Acer
SP300	3560	Laptop	Acer
Inspiron	3200	Desktop	Dell
MultiTouch	12400	Tablet	Hitachi

```
SELECT PNazwa AS Nazwa_produktu, Cena AS Cena_brutto, Producent
FROM   Produkt
WHERE  Kategoria = 'Laptop'
```

Nazwa_produktu	Cena_brutto	Producent
Aspire	2340	Acer
SP300	3560	Acer

Funkcje skalarne (1)

- UCASE() - konwersja małych liter na duże

```
SELECT UCASE(nazwa_kolumny) FROM nazwa_tabeli;
```

- LCASE() - konwersja dużych liter na małe

```
SELECT LCASE(nazwa_kolumny) FROM nazwa_tabeli;
```

- MID() - ekstrakcja fragmentu ciągu znaków

```
SELECT MID(nazwa_kolumny,początek,długość) FROM
nazwa_tabeli;
```

- LEN() - zwraca długość ciągu znaków

```
SELECT LENGTH(nazwa_kolumny) FROM nazwa_tabeli;
```

Funkcje skalarne (2)

- ROUND() - zaokrąglenie wartości numerycznej do podanej liczby znaków

```
SELECT ROUND(nazwa_kolumny,liczba_znaków) FROM nazwa_tabeli;
```

- NOW() - zwraca obecny czas i datę systemową

```
SELECT NOW() FROM nazwa_tabeli;
```

- FORMAT() - funkcja używana do określenia formatu wyświetlania wartości pola

```
SELECT FORMAT(nazwa_kolumny,format) FROM nazwa_tabeli;
SELECT NAZWISKO, FORMAT(Now(),'YYYY-MM-DD') AS Data FROM Studenci;
```

Funkcje agregujące (1)

- AVG() -zwraca średnią z wszystkich wartości danej kolumny

SELECT AVG(nazwa_kolumny) **FROM** nazwa_tabeli;

- COUNT() - zlicza wiersze zwracane przez SELECT

SELECT COUNT(nazwa_kolumny) **FROM** nazwa_tabeli;

- FIRST() - zwraca pierwszą wartość z wybranej kolumny

SELECT FIRST(nazwa_kolumny) **FROM** nazwa_tabeli;

- LAST() - zwraca ostatnią wartość z wybranej kolumny

SELECT LAST(nazwa_kolumny) **FROM** nazwa_tabeli;

Funkcje agregujące (2)

- MAX() - zwraca maksymalną wartość z wybranej kolumny

SELECT MAX(nazwa_kolumny) **FROM** nazwa_tabeli;

- MIN() - zwraca minimalną wartość z wybranej kolumny

SELECT MIN(nazwa_kolumny) **FROM** nazwa_tabeli;

- SUM() - zwraca sumę wszystkich wartości w kolumnie

SELECT SUM(nazwa_kolumny) **FROM** nazwa_tabeli;

ORDER BY: Sortowanie wyniku

```
SELECT    PNazwa, Cena, Producent
FROM      Produkt
WHERE     Kategoria='laptop' AND Cena > 50
ORDER BY  Cena, PNazwa
```

Limitowanie wyniku (LIMIT, OFFSET)

```
SELECT    PNazwa, Cena, Producent
FROM      Produkt
WHERE     Kategoria='laptop' AND Cena > 50
LIMIT 5;
```

Zwróci pierwsze 5 rekordów z całości wyniku

```
SELECT    PNazwa, Cena, Producent
FROM      Produkt
WHERE     Kategoria='laptop' AND Cena > 50
LIMIT 5 OFFSET 3;
```

Zwróci 5 rekordów licząc od trzeciego z całości wyniku

DISTINCT: Eliminacja duplikatów w wyniku

```
SELECT DISTINCT Kategoria
FROM   Produkt
```

Kategoria
Laptop
Desktop
Tablet

Versus

```
SELECT Kategoria
FROM   Product
```

Kategoria
Laptop
Laptop
Desktop
Tablet

Operacje na zbiorach i podzapytania

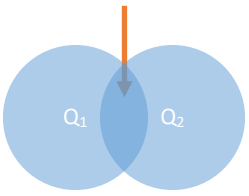
Operacje na zbiorach

INTERSECT - przecięcie, część wspólna

Nie działa w MySQL v.5.5.59

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
INTERSECT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$

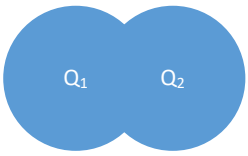


Eliminuje duplikaty

UNION - złączenie, suma

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$

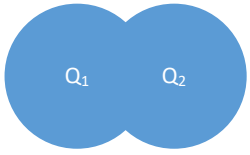


Eliminuje duplikaty

UNION ALL

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION ALL
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$



Nie eliminuje duplikatów

EXCEPT - różnica

Nie działa w MySQL v.5.5.59

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
EXCEPT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \setminus \{r.A \mid r.A = t.A\}$$

Eliminuje duplikaty

Reguły stosowania operatorów zbiorowych

- W łączonych operatorami zbiorowymi klauzulach SELECT musi wystąpić ta sama liczba atrybutów.
- Typy odpowiednich atrybutów różnych klauzul SELECT muszą być zgodne.
- W wyniku zapytania pojawiają się nazwy atrybutów wyłącznie z pierwszej klauzuli SELECT.
- Klauzula ORDER BY może być użyta tylko jako ostatnia klauzula zapytania.

Podzapytania

Podzapytanie

Mamy dany schemat:

```
Firma(nazwa, miasto)
Produkt(nazwa, producent)
Sprzedaz(id, produkt, kupujacy)
```

Jest to jeden ze sposobów uwzględnienia relacji w bazie danych

```
SELECT f.miasto
FROM   Firma f
WHERE  f.nazwa  IN (
     SELECT pr.producent
      FROM   Sprzedaz s, Produkt pr
      WHERE  s.produkt = pr.nazwa
        AND s.kupujacy = 'Jan Kowalski')
```

“Miasta, w których mieszczą się firmy wytwarzające produkty kupione przez Jana Kowalskiego”

Podzapytania

```
SELECT DISTINCT f.miasto
FROM   Firma f,
       Produkt pr,
       Sprzedaz s
WHERE  f.nazwa = pr.producent
  AND  pr.nazwa = s.produkt
  AND  s.kupujacy = 'Jak Kowalski'
```

```
SELECT DISTINCT f.miasto
FROM   Firma f
WHERE  f.nazwa  IN (
      SELECT pr.producent
      FROM   Sprzedaz s, Produkt pr
      WHERE  s.produkt = pr.nazwa
         AND s.kupujacy = 'Jan Kowalski')
```

Obie kwerendy są równoważne

Podzapytania jako alternatywa dla INTERSECT i EXCEPT

W niektóre SZBD nie obsługują INTERSECT i EXCEPT!

```
(SELECT R.A, R.B
 FROM   R)
INTERSECT
(SELECT S.A, S.B
 FROM   S)
```

⇨

```
SELECT R.A, R.B
FROM   R
WHERE EXISTS(
       SELECT *
       FROM S
        WHERE R.A=S.A AND R.B=S.B)
```

```
(SELECT R.A, R.B
 FROM   R)
EXCEPT
(SELECT S.A, S.B
 FROM   S)
```

⇨

```
SELECT R.A, R.B
FROM   R
WHERE NOT EXISTS(
       SELECT *
       FROM S
       WHERE R.A=S.A AND R.B=S.B)
```

Agregacja i grupowanie

Agregacja: COUNT

- COUNT zlicza wszystko w wyniku, łącznie z duplikatami

```
SELECT COUNT(kategoria)
FROM   Produkt
WHERE  rok > 1995
```

Zapewne chcielibyćmy otrzymać:

```
SELECT COUNT(DISTINCT kategoria)
FROM   Produkt
WHERE  rok > 1995
```

Więcej przykładów

```
Sprzedaz(produkt, data, cena, ilosc)
```

```
SELECT SUM(cena * ilosc)
FROM   Sprzedaz
```

Co to oznacza?

```
SELECT SUM(cena * ilosc)
FROM   Sprzedaz
WHERE  produkt = 'rogal'
```

Proste agregacje

Sprzedaz

Produkt	Data	Cena	Ilosc
rogal	10/21	1	20
banan	10/3	0.5	10
banan	10/10	1	10
rogal	10/25	1.50	20

```
SELECT SUM(cena * ilosc)
FROM   Sprzedaz
WHERE  produkt = 'rogal'
```

⇒ 50 (= 1∗20 + 1.50∗20)

Grupowanie i agregacja

```
Sprzedaz(produkt, data, cena, ilosc)
```

```
SELECT    produkt,
          SUM(cena * ilosc) AS CalSprzedaz
FROM      Sprzedaz
WHERE     data > '10/1/2005'
GROUP BY produkt
```

Znajdz całkowitą sprzedaż każdego produktu po 10/1/2005

Zobaczmy co to oznacza…

Grupowanie i agregacja

Semantyka zapytania:

1. Wykonaj klauzule FROM i WHERE

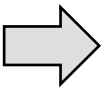
2. Pogrupuj według atrybutów stosując GROUP BY

3. Wykonaj klauzulę SELECT: na zgrupowanych atrybutach i zagregowanych wartościach

1. Wykonaj klauzule FROM i WHERE

```
SELECT   produkt, SUM(cena*ilosc) AS CalkSprzedaz
FROM     Sprzedaz
WHERE    data > '10/1/2005'
GROUP BY produkt
```

FROM



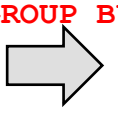
Produkt	Data	Cena	Ilosc
rogal	10/21	1	20
banan	10/3	0.5	10
banan	10/10	1	10
rogal	10/25	1.50	20

2. Pogrupuj według atrybutów stosując GROUP BY

```
SELECT   produkt, SUM(cena*ilosc) AS CalkSprzedaz
FROM     Sprzedaz
WHERE    data > '10/1/2005'
GROUP BY produkt
```

Produkt	Data	Cena	Ilosc
rogal	10/21	1	20
banan	10/3	0.5	10
banan	10/10	1	10
rogal	10/25	1.50	20

GROUP BY



Produkt	Data	Cena	Ilosc
rogal	10/21	1	20
	10/25	1.50	20
Banan	10/3	0.5	10
	10/10	1	10

3. Wykonaj klauzulę SELECT: na zgrupowanych atrybutach i zagregowanych wartościach

```
SELECT   produkt, SUM(cena*ilosc) AS CalkSprzedaz
FROM     Sprzedaz
WHERE    data > '10/1/2005'
GROUP BY produkt
```

Produkt	Data	Cena	Ilosc
rogal	10/21	1	20
	10/25	1.50	20
Banan	10/3	0.5	10
	10/10	1	10

SELECT

Produkt	CalkSprzedaz
Bagel	50
Banana	15

GROUP BY a podzapytania

```
SELECT   produkt,SUM(cena * ilosc) AS CalSprzedaz
FROM     Sprzedaz
WHERE    data > '10/1/2005'
GROUP BY produkt
```

```
SELECT DISTINCT x.produkt,
       (SELECT Sum(y.cena*y.ilosc)
        FROM   Sprzedaz y
        WHERE  x.produkt = y.produkt
           AND y.data > '10/1/2005') AS CalSprzedaz
FROM   Sprzedaz x
WHERE  x.data > '10/1/2005'
```

Klauzula HAVING

```
SELECT   produkt, SUM(cena*ilosc)
FROM     Sprzedaz
WHERE    data > '10/1/2005'
GROUP BY produkt
HAVING   SUM(ilosc) > 100
```

Takie samo zpytanie jak poprzednio z tym, że wybieramy tylko te produkty, które mają więcej jak 100 kupujących

HAVING uzywamy z wynikami **agregowanymi** (razem z funkcją aglegującą)

*Podczas gdy WHERE uzywamy z **pojedynczymi krotkami** (nieagregowanymi)*

Zapytania wielotabelowe

Klucz obcy

- Załóżmy, że mamy poniższy schemat:

```
Studenci(sid: string, imie: string, srednia: float)
Zapisani_studenci(student_id: string, idKursu: string, ocena: string)
```

- I chcemy wymusić następujące ograniczenie:
  - 'Tylko istniejący studenci mogą zapisać się na zajęcia' innymi słowy dane studenta muszą być w tabeli Studenci aby zapisać go na zajęcia

Studenci

sid	imie	srednia
101	Bob	3.2
123	Mary	3.8

Zapisani_studenci

student_id	idKursu	grade
123	564	A
123	537	A+

Mówimy, że student_id jest **kluczem obcym** (FK) odwołującym się do tabeli Studenci

Deklaracja kluczy obcych w SQL

```
Studenci(sid: string, imie: string, srednia: float)
Zapisani_studenci(student_id: string, idKursu: string, ocena: string)

CREATE TABLE Zapisani_studenci(
      student_id CHAR(20),
      idKursu          CHAR(20),
      ocena CHAR(10),
      PRIMARY KEY (student_id, idKursu),
      FOREIGN KEY (student_id) REFERENCES Students(sid)
);
```

Klucze obce i aktualizacja danych

```
Studenci(sid: string, imie: string, srednia: float)
Zapisani_studenci(student_id: string, idKursu: string, ocena: string)
```

- Co jeżeli chcemy wstawić wiersz do Zapisani_studenci bez powiązania z tabelą Student?
  - Operacja INSERT zostanie odrzucona (klucze obce są ograniczeniami)!

- Co jeżeli usuwamy studenta?
  1. Usunięcie zostanie zabronione, lub
  2. Usunięte zostaną wszystkie kursy usuwanego studenta

Złączenia (Joins) (1/4)

```
Produkt(PName, Cena, Kategoria, Producent)
Firma(FName, CenaAkcji, Kraj)
```

Przykład: Znajdź wszystkie produkty w cenie poniżej 4000zł wyprodukowane w Japonii; wyświetl nazwy i ceny.

```
SELECT PName, Cena
FROM   Produkt, Firma
WHERE  Producent = FName
       AND Kraj='Japan'
       AND Cena <= 4000
```

Złączenia (Joins) (2/4)

```
Produkt(PName, Cena, Kategoria, Producent)
Firma(FName, CenaAkcji, Kraj)
```

Przykład: Znajdź wszystkie produkty w cenie poniżej 4000zł wyprodukowane w Japonii; wyświetl nazwy i ceny.

```
SELECT PName, Cena
FROM   Produkt, Firma
WHERE  Producent = FName
       AND Kraj='Japan'
       AND Cena <= 4000
```

Złączenie między tabelami zwraca wszystkie unikalne kombinacje ich krotek, które **spełniają określone warunki złączenia**

Złączenia (Joins) (3/4)

```
Produkt(PName, Cena, Kategoria, Producent)
Firma(FName, CenaAkcji, Kraj)
```

Kilka równoznacznych sposobów definiowania prostego złączenia w SQL:

```
SELECT PName, Cena
FROM   Produkt, Firma
WHERE  Producent = FName
       AND Kraj='Japan'
       AND Cena <= 4000
```

```
SELECT PName, Cena
FROM   Produkt
JOIN   Firma ON Producent = Fname
               AND Kraj='Japan'
WHERE  Cena <= 4000
```

Złączenia (Joins) (4/4)

Produkt

PNazwa	Cena	Producent
Aspire	2340	Acer
SP300	3560	Acer
Satellite	3200	Toshiba
MultiTouch	12400	Hitachi

Firma

Fname	Cena Akcji	Kraj
Acer	25	USA
Toshiba	65	Japan
Hitachi	15	Japan

```
SELECT PName, Cena
FROM   Produkt, Firma
WHERE  Producent = FName
       AND Kraj='Japan'
       AND Cena <= 4000
```

PName	Cena
Satellite	3200

Wieloznaczność nazwy atrybutu w zapytaniach wielotabelowych (1/2)

```
Osoba(nazwisko, adres, miejscePracy)
Firma(nazwa, adres)
```

```
SELECT DISTINCT nazwisko, addres
FROM            Osoba, Firma
WHERE           miejscePracy = nazwa
```

Do jakiego "adresu" zapytanie się odwołuje?

Wieloznaczność nazwy atrybutu w zapytaniach wielotabelowych (2/2)

```
Osoba(nazwisko, adres, miejscePracy)
Firma(nazwa, adres)
```

Oba sposoby uniknięcia wieloznaczności są równoważne

```
SELECT DISTINCT Osoba.nazwisko, Osoba.adres
FROM            Osoba, Firma
WHERE           Osoba.miejscePracy = Firma.nazwa
```

```
SELECT DISTINCT o.nazwisko, o.adres
FROM            Osoba o, Firma f
WHERE           o.miejscePracy = f.nazwa
```

Pewne subtelności złączeń (1/2)

```
Produkt(PName, Cena, Kategoria, Producent)
Firma(FName, CenaAkcji, Kraj)
```

Znajdź wszystkie kraje, które wytwarzają produkty w kategorii 'laptop'.

```
SELECT Kraj
FROM   Produkt, Firma
WHERE  Producent=FName AND Kategoria='laptop'
```

Pewne subtelności złączeń (2/2)

Produkt

PNazwa	Cena	Kategoria	Producent
Aspire	2340	Laptop	Acer
SP300	3560	Laptop	Acer
Satellite	3200	Desktop	Toshiba
MultiTouch	12400	Tablet	Hitachi

Firma

Fname	CenaAkcji	Kraj
Acer	25	USA
Toshiba	65	Japan
Hitachi	15	Japan

```
SELECT Kraj
FROM   Produkt, Firma
WHERE  Producent=Fname
   AND Kategoria='Laptop'
```

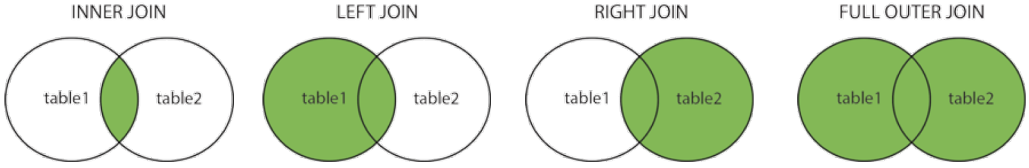
Kraj
USA
USA

Jakie jest rozwiązanie ?

Różne typy złączeń

- **(INNER) JOIN** - domyślne, zwraca wszystkie rekordy pasujące do lewej i prawej tabeli, jest to tzw. **złączenie wewnętrzne**
- **LEFT (OUTER) JOIN** - zwraca WSZYSTKIE rekordy z lewej i pasujące z prawej tabeli, jest to tzw. **złączenie zewnętrzne**.
- **RIGHT (OUTER) JOIN** - zwraca WSZYSTKIE rekordy z prawej i pasujące z lewej tabeli, jest to tzw. **złączenie zewnętrzne**.
- **FULL (OUTER) JOIN** - zwraca wszystkie rekordy z lewej i prawej zarówno pasujące jak i niepasujące, jest to tzw. **złączenie zewnętrzne**.

W **złączeniu zewnętrznym (OUTER)** pierwszy etap przebiega jak w domyślnym **JOIN**, następnie wynik jest uzupełniany wartościami **NULL** dla rekordów, które nie mają odpowiedników po drugiej stronie



Wyrażenie WITH, czyli tworzenie Common Table Expressions

Według dokumentacji MSDN (Microsoft Developer Network) Common Table Expression (CTE) to tymczasowy zbiór danych zdefiniowany w kontekście pojedynczej kwerendy SELECT, INSERT, UPDATE, DELETE i CREATE VIEW. CTE nie jest fizycznym obiektem w bazie danych i istnieje tylko w czasie wykonywania się jedenj operacji.

Wyrażenie WITH - podstawowa składnia

```
WITH
nazwa_podsumowania_danych AS (SELECT Wyrażnie)
   SELECT kolumny
   FROM nazwa_podsumowania_danych
   WHERE warunki <=> (
      SELECT kolumna
      FROM nazwa_podsumowania_danych)
   [ORDER BY kolumny]
```

Wyrażenie WITH - prosty przykład

```
WITH moi_klienci AS
(SELECT imie,nazwisko
FROM klienci
WHERE handlowiec = 'Janusz')
     SELECT * FROM moi_klienci;
```

Wyrażenie WITH - praktyczny przykład (1)

```
Firma(FName, CenaAkcji, Kraj)
Firma1(FName, CenaAkcji, Kraj)
```

Firma

FName	Cena Akcji	Kraj
Acer	25	USA
Toshiba	65	Japan
Hitachi	15	Japan
Lenovo	50	China
IBM	66	USA

Firma1

FName	Cena Akcji	Kraj
NULL	NULL	NULL

Wyrażenie WITH - praktyczny przykład (2)

```
WITH usuniete_wiersze AS (
DELETE FROM Firma
   WHERE
      CenaAkcji >= 50
   RETURNING *
)
INSERT INTO Firma1 (SELECT * FROM usuniete_wiersze)
```

Firma

FName	Cena Akcji	Kraj
Acer	25	USA
Hitachi	15	Japan

Firma1

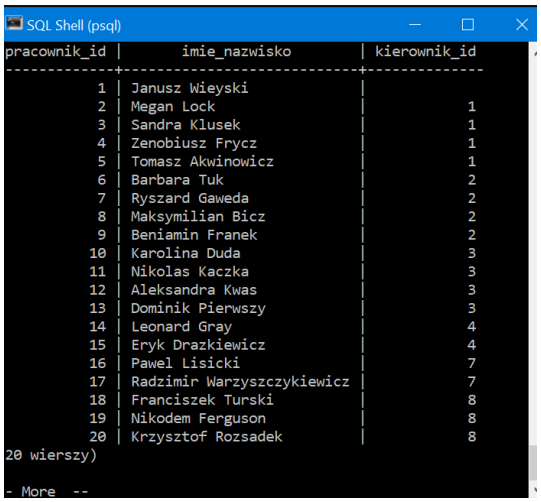
FName	Cena Akcji	Kraj
Toshiba	65	Japan
Lenovo	50	China
IBM	66	USA

Zapytania hierarchiczne

- **Zapytania hierarchiczne** pozwalają na rekurencję w relacjach zawierających dane hierarchiczne (np. szef – podwładny – podwładny podwładnego – …).
- Konieczna identyfikacja korzenia hierarchii (może być wiele korzeni – wiele hierarchii) oraz warunku wyszukującego rekordy-dzieci danego rekordu-rodzica.
- Realizacja **w standardzie SQL** – rekurencyjna **klauzula WITH**

Tabela z danymi w zależności hierarchicznej

```
Pracownicy(pracownik_id, imie_nazwisko, kierownik_id)
```



zobrazujmy hierarchię pracowników (przypiszmy poziomy organizacyjne poszczególnym pracownikom)

Zapytanie hierarchiczne

Wyrażenie nierekursywne

Wyrażenie rekursywne

```
WITH RECURSIVE podwladni AS (
SELECT
pracownik_id, kierownik_id,imie_nazwisko,0 AS Poziom_Org
FROM
pracownicy
WHERE
kierownik_id is NULL
UNION
SELECT
pr.pracownik_id,pr.kierownik_id,pr.imie_nazwisko,Poziom_Org + 1
FROM
pracownicy pr
INNER JOIN podwladni po ON po.pracownik_id = pr.kierownik_id
)
SELECT * FROM podwladni;
```

Jak to działa?

Wynik wyrażenia nierekursywnego:

SQL Shell (psql)

pracownik_id	kierownik_id	imie_nazwisko	poziom_org
1		Janusz Wieyski	0

(1 wiersz)

Wynik wyrażenia rekurencyjnego, jest rezultatem złączenia między tabelą **pracownicy** a obecnym stanem CTE **pracownicy.** Będzie on wejściem w następnej iteracji wyrażenia rekurencyjnego.

SQL Shell (psql)

pracownik_id	kierownik_id	imie_nazwisko	poziom_org
2	1	Megan Lock	1
3	1	Sandra Klusek	1
4	1	Zenobiusz Frycz	1
5	1	Tomasz Akwinowicz	1

Jak to działa?

Wyniki kolejnych iteracji, na koniec zostaną złączone w jeden wynik.

SQL Shell (psql)

pracownik_id	kierownik_id	imie_nazwisko	poziom_org
6	2	Barbara Tuk	2
7	2	Ryszard Gaweda	2
8	2	Maksymilian Bicz	2
9	2	Beniamin Franek	2
10	3	Karolina Duda	2
11	3	Nikolas Kaczka	2
12	3	Aleksandra Kwas	2
13	3	Dominik Pierwszy	2
14	4	Leonard Gray	2
15	4	Eryk Drazkiewicz	2

SQL Shell (psql)

pracownik_id	kierownik_id	imie_nazwisko	poziom_org
16	7	Pawel Lisicki	3
17	7	Radzimir Warzyszczykiewicz	3
18	8	Franciszek Turski	3
19	8	Nikodem Ferguson	3
20	8	Krzysztof Rozsadek	3

Wynik zapytania hierarchicznego

SQL Shell (psql)

pracownik_id	kierownik_id	imie_nazwisko	poziom_org
1		Janusz Wieyski	0
2	1	Megan Lock	1
3	1	Sandra Klusek	1
4	1	Zenobiusz Frycz	1
5	1	Tomasz Akwinowicz	1
6	2	Barbara Tuk	2
7	2	Ryszard Gaweda	2
8	2	Maksymilian Bicz	2
9	2	Beniamin Franek	2
10	3	Karolina Duda	2
11	3	Nikolas Kaczka	2
12	3	Aleksandra Kwas	2
13	3	Dominik Pierwszy	2
14	4	Leonard Gray	2
15	4	Eryk Drazkiewicz	2
16	7	Pawel Lisicki	3
17	7	Radzimir Warzyszczykiewicz	3
18	8	Franciszek Turski	3
19	8	Nikodem Ferguson	3
20	8	Krzysztof Rozsadek	3

(20 wierszy)

Manipulowanie danymi

Tworzenie i usuwanie tabel

```
CREATE TABLE nazwa_tabeli(nazwa_pola1 typ, nazwa_pola2, typ,
inne_parametry, ...)

CREATE TABLE firma (nazwa char(20), miasto char(20));

DROP TABLE nazwa_tabeli;
```

Wstawianie i usuwanie danych

```
INSERT INTO nazwa_tabeli VALUES(wartości)

INSERT INTO firma VALUES("Compaq","Olsztyn");

INSERT INTO firma(miasto) VALUES("Warszawa");

DELETE FROM firma;
DELETE FROM firma WHERE nazwa="Compaq";
```

Aktualizacja danych w tabeli

```
UPDATE nazwa_tabeli SET pole=wartość
     WHERE warunek;

UPDATE firma SET name="Panasonic"
     WHERE miasto="Warszawa";
```

Aktualizacja struktury tabeli (dodaj/usuń kolumnę)

```
ALTER TABLE nazwa_tabeli operacja ADD lub DROP
```

Dodaj kolumnę “adres”

```
ALTER TABLE firma ADD adres char(20);
```

Usuń kolumnę “adres”

```
ALTER TABLE company DROP COLUMN address;
```

Dodaj więcej niż jedną kolumnę

```
ALTER TABLE company ADD address char(20), ADD value
int(5);
```

Widoki

- Widok (ang View) to „wirtualna” tabela, którą tworzymy za pośrednictwem zapytania.
- Z widoku korzystamy jak ze zwykłej tabeli, możemy więc wykonywać „na nim” dowolne kwerendy.
- Od strony SZBD widok różni się od tabeli tym, że do póki nie zostanie utworzony dla nim indeks nie istnieje fizyczna reprezentacja jego danych poza pamiącią operacyjną.
- Widok może wydzielać fragment danych, który będzie udostępniony użytkownikowi

Tworzenie i usuwanie widoków

Widok ze wszystkich pól tabeli klienci:

```
CREATE VIEW moj_widok AS SELECT * FROM klienci;
```

Wybranie danych z utworzonego widoku

```
SELECT * FROM moj_widok;
```

Indeksy w bazie danych

- Indeks jest specjalną strukturą danych wprowadzoną w celu zwiększenia szybkości wykonywania operacji na tabeli.
- W uproszczeniu można powiedzieć, że indeks w bazie danych jest odpowiednikiem spisu treści w książce.
- Indeks jest plikiem zawierającym pary kluczy i wskaźników dla każdego określonego fragmentu danych (większego lub mniejszego w zależności od rodzaju indeksu) w pliku zawierającym dane. Każdy klucz w tym pliku jest powiązany z poszczególnym wskaźnikiem do rekordu w pliku z sortowanymi danymi.
- Oznacza to, że ze stosowaniem indeksów obok wzrostu szybkości wyszukiwania należy wziąć pod uwagę wzrost zajętości przestrzeni dyskowej.

Do których kolumn należy tworzyć indeksy?

- Selektywność indeksu - parametr określający jego przydatność:

$$S = \frac{U}{W}$$

- U - liczba unikalnych wartości dla kolumny
- W - liczba wszystkich wierszy w kolumnie

Serwer zazwyczaj nie będzie korzystał z indeksu o selektywności mniejszej niż 85%

Do których kolumn należy tworzyć indeksy?

W praktyce tworzymy indeksy dla:

- Kolumn z ograniczeniem PRIMARY KEY.
- Kolumn z ograniczeniem FOREIGN KEY oraz kolumn wykorzystywanych przy łączeniu tabel.
- Kolumn przechowujących dane wykorzystywane jako argument wyszukiwania.
- Kolumn przechowujących często sortowane dane

Tworzenie indeksu - przykłady

Podczas tworzenia tabeli

```
CREATE TABLE nazwa_tabeli (
  kolumna1 INT,
  kolumna2 INT,
  UNIQUE indeks_unikalny (kolumna1),
  INDEX indeks_zwykly (kolumna2)
);
```

W istniejącej tabeli używając ALTER TABLE

```
ALTER TABLE nazwa_tabeli
ADD UNIQUE indeks_unikalny (kolumna1),
ADD INDEX indeks_zwykly (kolumna2)
```

W istniejącej tabeli używając CREATE

```
CREATE UNIQUE INDEX indeks_unikalny ON nazwa_tabeli (kolumna1)
CREATE INDEX indeks_zwykly ON nazwa_tabeli (kolumna2)
CREATE INDEX indeks_imie_nazw ON osoby (imie, nazwisko)
```

Transakcje

Transakcja może zawierać jedną operację

```
START TRANSACTION
      UPDATE Produkt
      SET Cena = Cena - 1.99
      WHERE prod_nazwa = 'chleb'
COMMIT
```

Polecenie **COMMIT** zatwierdza działanie transakcji.
Wynik można cofnąć używając **ROLLBACK**, lecz **zanim użyje się COMMIT!**

Lub dowolną ilość dowolnych operacji

```
START TRANSACTION
      UPDATE Bank SET amount = amount - 100
      WHERE name = 'Bob'
      UPDATE Bank SET amount = amount + 100
      WHERE name = 'Joe'
COMMIT
```

Transakcje - jak zwykły użytkownik bazy może z nich korzystać?

Użytkownik tworzy tabelę

```
mysql> CREATE TABLE tab (f INT) TYPE=InnoDB;
```

Rozpoczyna transakcję i wstawia nowy rekord

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tab(f) VALUES (1);
Query OK, 1 row affected (0.01 sec)
```

Sprawdza zawartość tabeli

```
mysql> SELECT * FROM tab;
+---+
| f |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

Transakcje - jak zwykły użytkownik bazy może z nich korzystać?

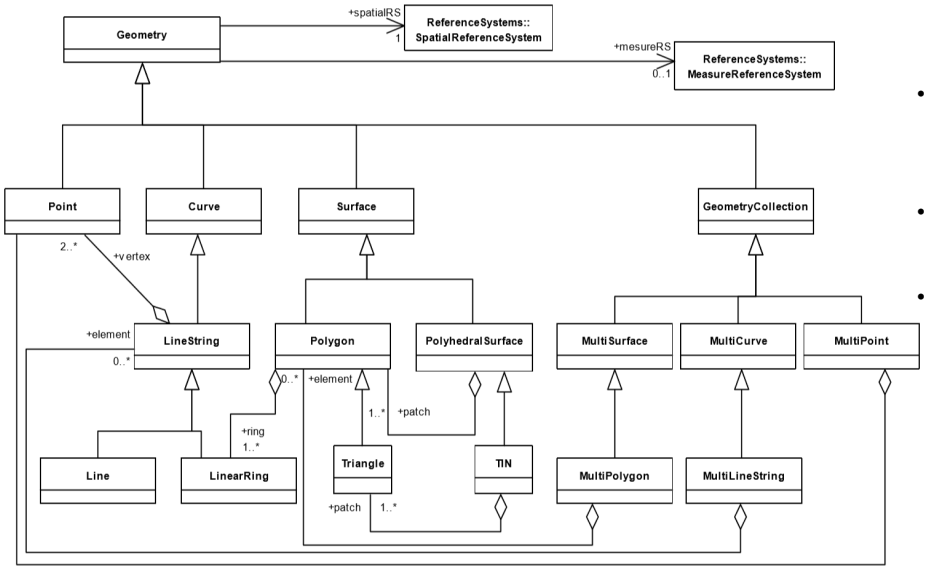
Jeżli użyłby polecenia COMMIT, dane zostaną zapamiętane w bazie, jeżeli natomiast użyje ROLLBACK:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

Zmiany zostaną wycofane

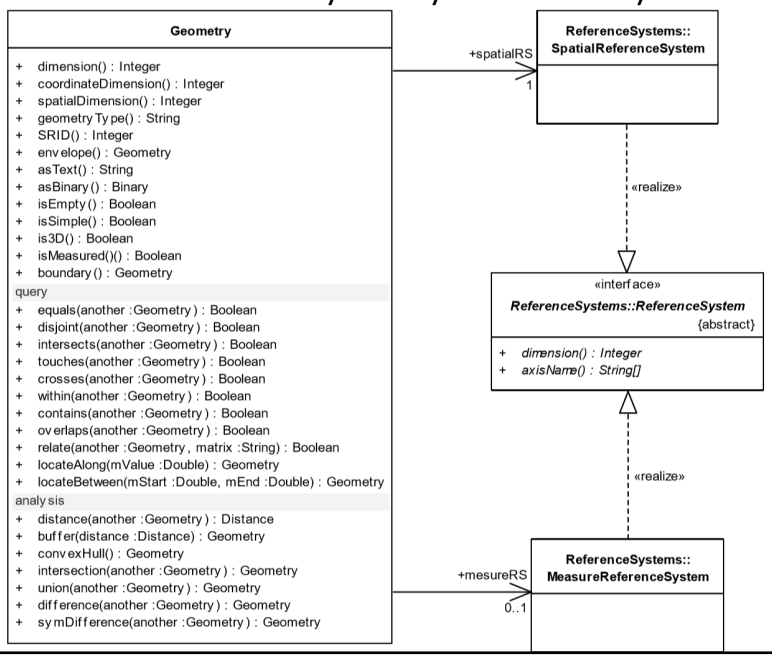
```
mysql> SELECT * FROM tab;
Empty set (0.00 sec)
```

Model obiektowy Simple feature (OGC)



- Model obiektowy Simple Features jest niezależny od platformy, zapisany jest w notacji UML.
- Klasą główną jest Geometry, z której dziedziczą Point, Curve, Surface i GeometryCollection.
- Każdy obiekt geometryczny jest powiązany z systemem odniesień przestrzennych (SRS).

Pola i metody klasy Geometry



Jest to **nadrzędna klasa**, zawiera większość właściwości. Ponadto **każda klasa wywiedziona** z Geometry może posiadać **własne, specyficzne metody i pola.**

Reprezentacja geometrii w bazie danych lub pliku

- Well Known Text Representation (WKT)
- Well Known Binary Representation (WKB)

Well Known Text Representation - w przykładach

- Każdy typ geometryczny posiada reprezentację WKT, która może być użyta zarówno do tworzenia nowych geometrii danego typu jak i przekształcania istniejących instancji do formy tekstowej aby wyświetlić ją w postaci alfanumerycznej.

Geometry Type	Text Literal Representation	Comment
Point	Point (10 10)	a Point
LineString	LineString (10 10, 20 20, 30 40)	a LineString with 3 points
Polygon	Polygon ((10 10, 10 20, 20 20, 20 15, 10 10))	a Polygon with 1 exteriorRing and 0 interiorRings
Multipoint	MultiPoint ((10 10), (20 20))	a MultiPoint with 2 points
MultiLineString	MultiLineString ((10 10, 20 20), (15 15, 30 15))	a MultiLineString with 2 linestrings
MultiPolygon	MultiPolygon (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))	a MultiPolygon with 2 polygons

Well Known Text Representation - w przykładach

Geometry Type	Text Literal Representation	Comment
Tin	Tin Z (((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 0 0 1, 0 0 0)), ((1 0 0, 0 1 0, 0 0 1, 1 0 0)),)	A tetrahedron (4 triangular faces), corner at the origin and each unit coordinate digit.
Point	Point Z (10 10 5)	a 3D Point
Point	Point ZM (10 10 5 40)	the same 3D Point with M value of 40
Point	Point M (10 10 40)	a 2D Point with M value of 40
Polyhedron	Polyhedron Z (((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1. 0 0 1)))	A polyhedron cube, corner at the origin and opposite corner at (1, 1, 1).

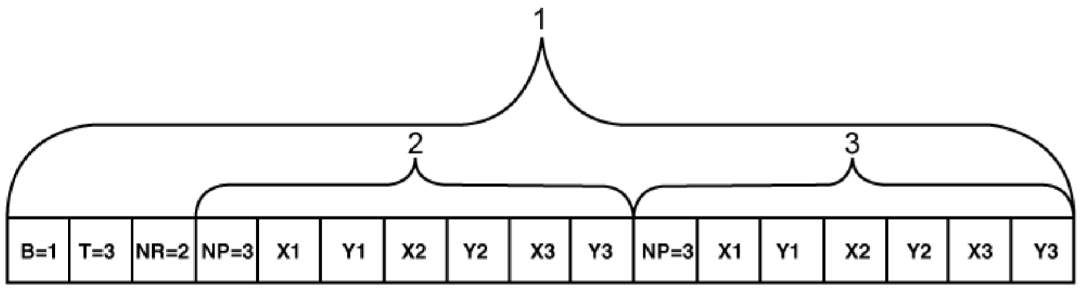
Well Known Binary Representation

- Reprezentacja WKB jest zapisem geometrii obiektu w postaci strumienia bajtów. Umożliwia wymianę obiektów geometrycznych w bazie danych między klientem a serwerem w formie binarnej.
- WKB serializuje obiekt geometryczny jako sekwencję typów numerycznych ze zbioru {Unsigned Integer, Double}. Typy te następnie są serializowane jako sekwencje bajtów z wykorzystaniem jednego z dwóch standardów dla typów numerycznych (NDR lub XDR)
- Enkodowanie NDR lub XDR oznacza kolejność bajtów w strumieniu:
  - XDR enkodowanie Big Endian (bajt najbardziej znaczący jako pierwszy {double} lub bit znaku pierwszy {int.})
  - NDR encodowanie Little Endian (bajt najmniej znaczący jako pierwszy {double} lub bit znaku ostatni {int.})

Przykładowe kody integer dla typów geometrycznych

Type	Code
Geometry	0
Point	1
LineString	2
Polygon	3
MultiPoint	4
MultiLineString	5
MultiPolygon	6
GeometryCollection	7
CircularString	8
CompoundCurve	9
CurvePolygon	10
MultiCurve	11
MultiSurface	12
Curve	13
Surface	14
PolyhedralSurface	15
TIN	16

Przykładowa reprezentacja WKB obiektu geometrycznego



Key

1 WKB Polygon
2 ring 1
3 ring 2

Figure 25: Well-known Binary Representation for a geometric object in NDR format (B = 1) of type Polygon (T = 3) with 2 LinearRings (NR = 2) each LinearRing having 3 points (NP = 3)

Tworzenie tabeli z danymi przestrzennymi

```
CREATE TABLE buildings (
   fid       INTEGER NOT NULL PRIMARY KEY,
   addressCHARACTER VARYING(64),
   position     POINT,
   footprint    POLYGON);
```

Funkcje tworzące wartości geometryczne na bazie reprezentacji WKT

Funkcje te przyjmują argument w postaci reprezentacji WKT, oraz opcjonalnie identyfikatora systemu odniesień przestrzennych (SRID). Zwracają odpowiedni rodzaj geometrii.

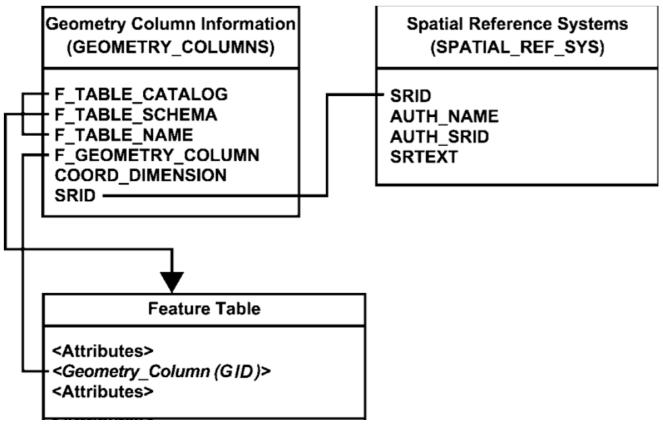
```
GeomFromText(WKT[,srid]),
LineFromText(WKT[,srid]),
PointFromText(WKT[,srid]),
PolyFromText(WKT[,srid]),
...
```

Funkcje tworzące wartości geometryczne na bazie reprezentacji WKB

Działające analogicznie do poprzednich. Wejściem jest reprezentacja WKB:

```
GeomFromWKB(WKT[,srid]),
LineFromWKB(WKT[,srid]),
PointFromWKB(WKT[,srid]),
PolyFromWKB(WKT[,srid]),
...
```

Architektura OGC — implementacja SQL z wykorzystaniem typów geometrycznych



a) Tabela **GEOMETRY_COLUMNS** opisuje dostępne tabele obiektów i ich własności geometryczne. W Postgre od wersji 2.0.0. jest to widok (nie tabela).
b) Tabela **SPATIAL_REF_SYS** opisuje System odniesień przestrzennych dla geometrii.
c) Tabela **Feature Table** przechowuje zbiór obiektów (feature). jej kolumny reprezentują atrybuty obiektów, natomiast wiercze odpowiadają poszczególnym instancjom obiektów. Atrybut Geometry jest jednym z atrybutów (kolumn) i opisuje geometrię obiektu.

Wstawianie danych wektorowych do tabeli

```
INSERT INTO buildings VALUES(113, '123 Main Street',
   PointFromText(
      'POINT( 52 30 )', 101),
   PolyFromText(
      'POLYGON( ( 50 31, 54 31, 54 29, 50 29, 50 31) )', 101));
```

Wybieranie danych wektorowych z tabeli

Dane przestrzenne z bazy danych wybieramy używając zapytania SQL rozszerzonego o funkcje SQL do tworzenia i przekształcania typów geometrycznych, zgodnie ze specyfikacją SQL/MM

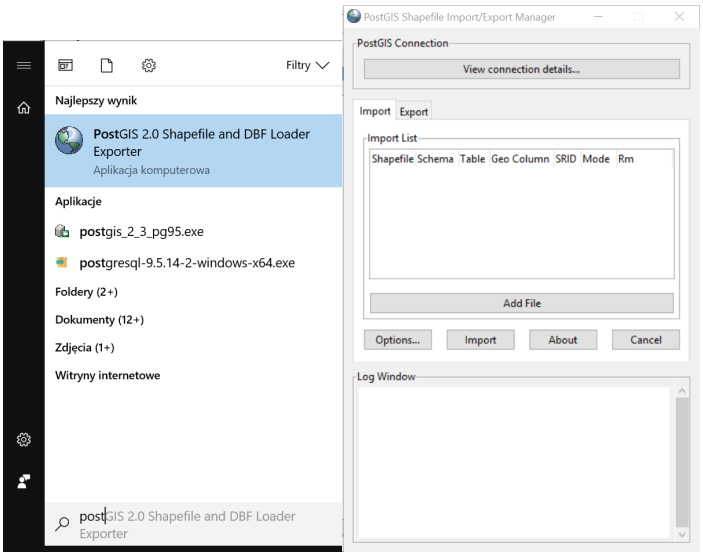
```
SELECT AsText(position)
  FROM buildings
  WHERE address = '123 Main Street';
```

Funkcje operujące na danych przestrzennych

Standard OGC definiuje wiele funkcji do wykonywania analiz przestrzennych np.:

- Area()
- Centroid()
- Contains()
- Buffer()
- Crosses()
- Disjoint()
- Dimension()
- Equals()
- ... i wiele innych....

Wczytywanie danych wektorowych - narzędzie Loader



Operatory w PostGIS

- W PostGIS zdefiniowano zestaw operatorów do wykonywania operacji na danych geometrycznych i MBR obiektów, np:
- && - zwraca TRUE jeżeli MBR jednej geometrii przecina MBR drugiej

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
      (1, 'LINESTRING(0 0, 3 3)'::geometry),
      (2, 'LINESTRING(0 1, 0 5)'::geometry)) AS tbl1,
( VALUES
      (3, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl2;

 column1 | column1 | overlaps
---------+---------+----------
         1 |       3 | t
         2 |       3 | f
```

Operatory w PostGIS

- A ~= B - zwraca TRUE jeżeli MBR obiektów A i B są takie same
- A <-> B - zwraca odległość między obiektami Ai B
- A ~ B - zwraca TRUE jeżeli MBR obiektu A zawiera MBR obiektu B
- A = B - zwraca TRUE, jeżeli współrzędne oraz ich kolejność w obiekcie A są tkie same jak współrzędne i ich kolejność w obiekcie B

- Więcej operatorów wraz z przykładami:

https://postgis.net/docs/reference.html#Operators

Funkcje wektorowe w PostGIS

- PostGIS posiada bardzo bogatą kolekcję funkcji do przetwarzania i analizy danych przestrzennych. Możemy je podzielić na:
- Konstruktory geometrii
  - ST_GeomFromText, ST_GeomFromWKB, ST_MakePoint, ST_PolygonFromText i inne.
- Funkcje dostępu do danych
  - ST_Boundary, ST_Envelope, ST_EndPoint i inne.
- Edytory geometrii
  - ST_AddPoint, ST_Affine, ST_Rotate, ST_LineMerge i inne,
- Formatujące wyjście
  - ST_AsBinary, ST_AsText, ST_AsGML, ST_AsGeoJSON i inne,
- Relacji przestrzennych i pomiarów
  - ST_Distance, ST_Disjoint, ST_Intersects, ST_Length, ST_Within i inne.

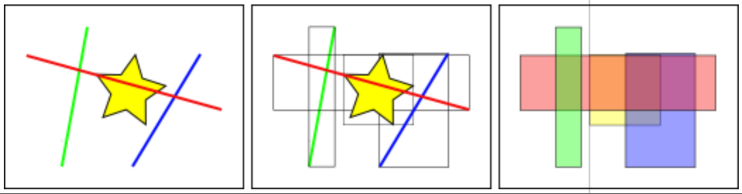
https://postgis.net/docs/reference.html

Indeksy przestrzenne

- Stosowanie indeksów przestrzennych przyspiesza operacje na danych geometrycznych (np. analizy przestrzenne)

```
CREATE INDEX ulice_gix ON ulice USING GIST (the_geom);
```

- Powyższy przykład tworzy indeks '**ulice_gix**' na kolumnie geometrycznej '**the_geom**' w tabeli '**ulice**'
- słowo kluczowe **USING GIST** określa rodzaj indeksu (GIST). Standardowy indeks tworzy drzewo B-tree, co wymagałoby indeksacji wszystkich wierzchołków. GIST tworzy **prostokąty MBR**, co w przypadku danych geometrycznych jest bardziej wydajne i zajmuje mniej pamięci.



Wczytanie rastra z pliku

Narzędzie konsolowe

raster2pgsql opcje_rastra plik_rastra schematPG.tabelaPG > out.sql

Przykład wywołania

raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation > elev.sql

psql -d gisdb -f elev.sql

Opis:
https://postgis.net/docs/using_raster_dataman.html#RT_Raster_Loader

Funkcje zarządzające danymi rastrowymi i operatory rastrowe

- Tabela rastrowa

```
CREATE TABLE myRasterTable(rid serial primary key, rast raster);
```

- Wstawienie danych rastrowych

```
INSERT INTO myRasterTable(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
```

- Stworzenie kolejnej tabeli na bazie istniejących danych rastrowych

```
INSERT INTO myRasterTable(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
  FROM myRasterTable WHERE rid = 3;
```

Funkcje zarządzające danymi rastrowymi i operatory rastrowe

- Wyświetlenie zawtości tabeli rastrowej

```
SELECT rid, (md).*
  FROM (SELECT rid, ST_MetaData(rast) As md
              FROM myRasterTable
              WHERE rid IN(3,4)) As foo;
```

- Stworzenie rastra z istniejącej geometrii wektorowej

```
CREATE TABLE myNewRaster AS
  SELECT 1 AS rid, ST_AsRaster((
       SELECT
          ST_Collect(geom)
       FROM myGeomTable
       ), 1000.0, 1000.0 )
  AS rast;
```

MINISTERSTWO ŚRODOWISKA

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej

TRAPPA

Szkolenie
Język SQL w bazie PostgreSQL

Dziękuję za uwagę

Sfinansowano ze środków Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej