Bazy danych
Podstawy relacyjnych baz danych

Marcin Szpyrka

Katedra Informatyki Stosowanej
AGH w Krakowie

2015/16

Literatura

1. Jeffrey D. Ullman, Jennifer Widom: *Podstawowy kurs systemów baz danych*, Helion, Gliwice, 2011
2. Thomas Connolly, Carolyn Begg: *Systemy baz danych*, tom 1 i 2, Wydawnictwo RM, Warszawa, 2004
3. Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: *Systemy baz danych. Pełny wykład*, WNT, Warszawa, 2006
4. Chris J. Date: *Relacyjne bazy danych dla praktyków*, Wydawnictwo Helion, Gliwice, 2006
5. Joe Celko: *SQL zaawansowane techniki programowania*, Wydawnictwo Naukowe PWN, Warszawa, 2008
6. Sharon Allen: *Modelowanie danych*, Wydawnictwo Helion, Gliwice, 2006
7. http://www.postgresql.org – dokumentacja systemu PostgreSQL.
8. Antoni Ligęza: Materiały do wykładów z baz danych, http://home.agh.edu.pl/ligeza/wiki

Informacja, dane

Informacją nazywamy wielkość abstrakcyjną, która może być przechowywana w pewnych obiektach, przesyłana między pewnymi obiektami, przetwarzana w pewnych obiektach i stosowana do sterowania pewnymi obiektami, przy czym przez obiekty rozumie się organizmy żywe, urządzenia techniczne oraz systemy takich obiektów.
Dane jest to konkretna reprezentacja informacji.

- Wybór reprezentacji informacji jest bardzo ważny dla wygody przetwarzania danych, np. operacje na liczbach zespolonych w zależności od formy ich zapisu, operacje arytmetyczne na liczbach zapisanych cyframi rzymskimi.
- Dane przechowywane w bazie powinna cechować trwałość – dane mają być przechowywane przez pewien okres czasu, na ogół nieokreślony z góry; oraz zgodność z rzeczywistością – dane w bazie danych muszą stanowić wierne odzwierciedlenie modelowanego fragmentu rzeczywistości, a więc w miarę zachodzących zmian we fragmencie rzeczywistości, baza danych też musi się odpowiednio zmieniać.

Baza danych – definicje

(znaczenie potoczne) Baza danych są to dane oraz program komputerowy wyspecjalizowany do gromadzenia i przetwarzania tych danych.

(znaczenie słownikowe) Baza danych jest to zbiór powiązanych ze sobą informacji zorganizowanych w strukturę pozwalającą na łatwe ich przetwarzanie.

(art. 2 ustawy o ochronie baz danych z 9 listopada 2001 r.) Baza danych jest to zbiór danych lub jakichkolwiek innych materiałów i elementów zgromadzonych według określonej systematyki lub metody, indywidualnie dostępnych w jakikolwiek sposób, w tym środkami elektronicznymi, wymagający istotnego, co do jakości lub ilości, nakładu inwestycyjnego w celu sporządzenia, weryfikacji lub prezentacji jego zawartości

Podsumowanie: Baza danych dotyczy zawsze pewnego fragmentu rzeczywistości związanego z firmą, organizacją lub innym działającym systemem. Stanowi ona zbiór danych, posiadający określoną strukturę wewnętrzną, których zadaniem jest reprezentowanie tego fragmentu rzeczywistości i ułatwienie przetwarzania tych danych. Bazy danych tworzy się w celu efektywnego wykorzystywania dużych ilości danych.

System zarządzania bazą danych

Do zapewnienia obsługi bazy danych (rozumianej jako zbiór danych) wykorzystuje się system zarządzania bazą danych (SZBD) (ang. Database Management System, DBMS). Jest on zorganizowanym zbiorem narzędzi umożliwiających realizację istotnych dla użytkownika operacji na danych.

Przykłady: PostgreSQL, MySQL, Oracle, IBM DB2, Sybase, MS SQL Server.

- pierwsza generacja – systemy hierarchiczne – Pierwszym SZBD był System Apollo (lata sześćdziesiąte XX w.), który powstał na potrzeby zadania związanego z lotem człowieka na księżyc.
- druga generacja – systemy relacyjne – Początki tych systemów związane są opublikowaną w 1970 r. pracą E.F. Codda.
- trzecia generacja – systemy obiektowe i obiektowo-relacyjne

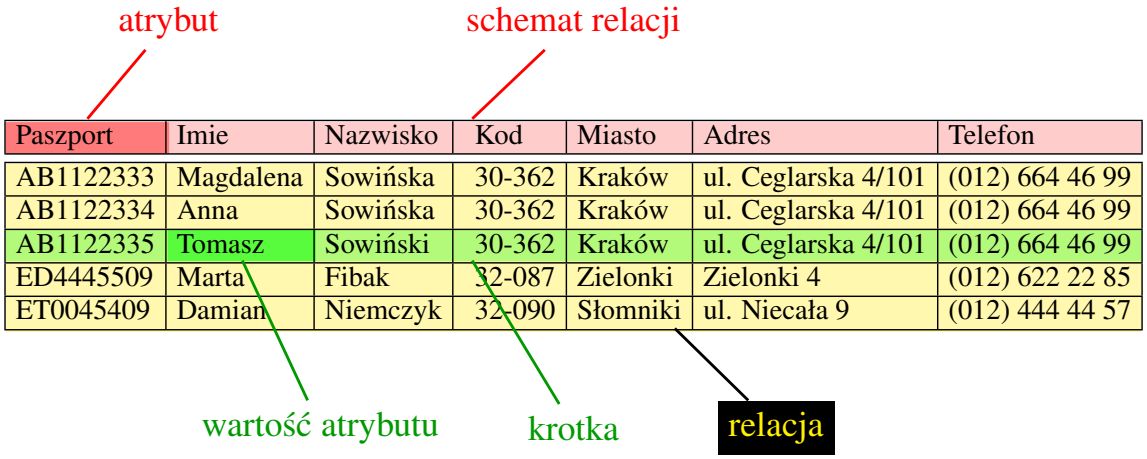
Edgar Frank „Ted" Codd (23.08.1923 – 18.04.2003) – brytyjski informatyk, główny twórca teorii relacyjnych baz danych, w 1981 r. otrzymał Nagrodę Turinga.

System zarządzania bazą danych – funkcje

1. Umożliwienie projektowania i implementacji nowej bazy przy użyciu narzędzi i języka definicji danych (ang. Data Definition Language, DDL).
2. Umożliwienie selektywnego dostępu do danych za pomocą języka zapytań i tworzonych w nim kwerend.
3. Umożliwienie wykonywania określonych operacji na danych przy pomocy języka operowania na danych (ang. Data Manipulation Language, DML).
4. Obsługa przechowywania dużych zbiorów danych, zapewnienie niezawodności oraz efektywności, obsługa operacji na plikach danych na dysku.
5. Zapewnienie integralności danych (na poziomie tabeli, na poziomie bazy danych).
6. Ochrona dostępu do danych, zapewnienie różnych obszarów i poziomów dostępu (utrzymywanie kont użytkowników, przypisywanie uprawnień).
7. Zapewnienie dostępu dla wielu użytkowników (wielodostępu) oraz synchronizacja dostępu w przypadku dostępu współbieżnego.
8. Zapewnienie możliwości komunikacji z innymi systemami.
9. Dostarczanie opisu i dokumentacji (schematu i struktury).
10. Optymalizacja pracy (minimalizacja czasu dostępu lub obsługi żądań, optymalizacja dostępu dla poszczególnych użytkowników (perspektywy), optymalizacja gospodarki zasobami i organizacji bazy danych.

Relacyjne bazy danych

Zgodnie z relacyjnym modelem danych baza danych jest zbiorem relacji (w sensie algebraicznym). Przyjmuje się powszechnie reprezentację, przy której pojedyncza relacja jest dwuwymiarową tabelą złożoną z kolumn i wierszy.



Paszport	Imie	Nazwisko	Kod	Miasto	Adres	Telefon
AB1122333	Magdalena	Sowińska	30-362	Kraków	ul. Ceglarska 4/101	(012) 664 46 99
AB1122334	Anna	Sowińska	30-362	Kraków	ul. Ceglarska 4/101	(012) 664 46 99
AB1122335	Tomasz	Sowiński	30-362	Kraków	ul. Ceglarska 4/101	(012) 664 46 99
ED4445509	Marta	Fibak	32-087	Zielonki	Zielonki 4	(012) 622 22 85
ET0045409	Damian	Niemczyk	32-090	Słomniki	ul. Niecała 9	(012) 444 44 57

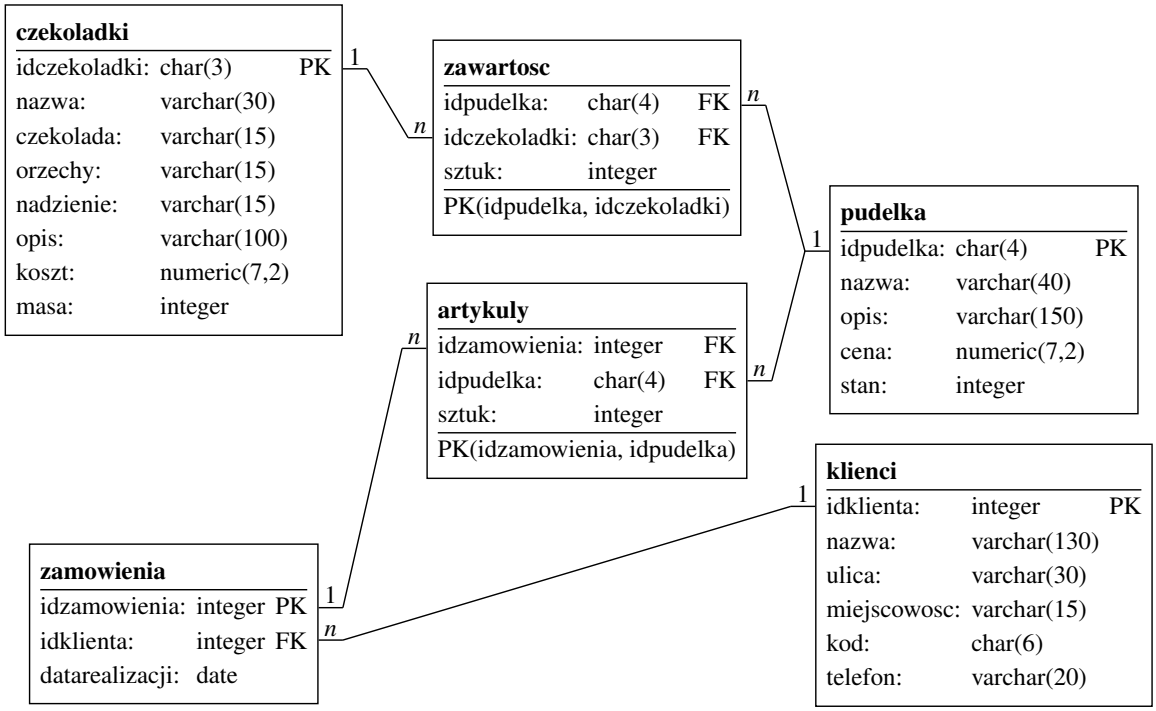
Podstawowe pojęcia

Paszport	Imie	Nazwisko	Kod	Miasto	Adres	Telefon
AB1122333	Magdalena	Sowińska	30-362	Kraków	ul. Ceglarska 4/101	(012) 664 46 99
AB1122334	Anna	Sowińska	30-362	Kraków	ul. Ceglarska 4/101	(012) 664 46 99
AB1122335	Tomasz	Sowiński	30-362	Kraków	ul. Ceglarska 4/101	(012) 664 46 99
ED4445509	Marta	Fibak	32-087	Zielonki	Zielonki 4	(012) 622 22 85
ET0045409	Damian	Niemczyk	32-090	Słomniki	ul. Niecała 9	(012) 444 44 57

- Atrybut reprezentuje pewną własność, cechę lub charakterystykę obiektu, pozwalającą częściowo opisać ten obiekt.
- Schemat relacji jest zbiorem atrybutów danej relacji – **Nie jest istotna kolejność atrybutów w schemacie relacji**, ale dla ustalenia uwagi przyjmujemy pewną domyślną kolejność atrybutów.
- Relacja jest zbiorem krotek – **Nie jest istotna kolejność krotek. Krotki nie powtarzają się.**
- Relacja jest **wartością** zmiennej relacyjnej. Operacje typu dodanie krotek, usunięcie, modyfikacja zmieniają tę wartość.

Określenia *atrybut, krotka, wartość atrybutu, zmienna relacyjna* są terminami modelu relacyjnego. W implementacjach często używa się terminów odpowiednio: *kolumna, rekord, pole rekordu, tabela*.

Baza danych *Cukiernia*



Dziedziny atrybutów

Z każdym atrybutem powiązana jest jego dziedzina, określająca dopuszczalne wartości tego atrybutu. W praktyce często dziedzinę atrybutu definiuje się poprzez podanie: typu danych, rozmiaru pola i więzów spójności.

Rodzaje dziedzin:

- binarne – np. wartości logiczne tak oraz nie;
- trójwartościowe – np. {TRUE, FALSE, NULL};
- nieuporządkowane – są to skończone (lub rzadziej nieskończone) zbiory nazw, które nie są powiązane ze sobą żadną relacją porządkującą (choć zazwyczaj dostępny jest porządek leksykograficzny);
- częściowo uporządkowane, uporządkowane – j.w., ale z relacją częściowego porządku lub relacją porządku;
- liczbowe – należą do dziedzin uporządkowanych, ale wyróżnia je określona arytmetyka pozwalająca realizować operacje obliczeniowe, np. podzbiory zbiorów $\mathbb{N}, \mathbb{Z}, \mathbb{R}$, daty, czas, typy walutowe itp.
- specjalizowane – tekst (bez porządku leksykograficznego), dźwięk, obraz, sekwencje wideo, hiperłącza itp.

Baza danych *Cukiernia* – dziedziny atrybutów

czekoladki		
idczekoladki:	char(3)	PK
nazwa:	varchar(30)	
czekolada:	varchar(15)	
orzechy:	varchar(15)	
nadzienie:	varchar(15)	
opis:	varchar(100)	
koszt:	numeric(7,2)	
masa:	integer	

zamowienia		
idzamowienia:	integer	PK
idpudelka:	integer	FK
datarealizacji:	date	

- char(n) – tekst o długości dokładnie *n* znaków (ewentualne uzupełnianie spacjami);
- varchar(n) – tekst o długości maksymalnie *n* znaków;
- integer – liczby całkowite (typ 4-bajtowy ze znakiem);
- numeric(m,n) – typ stałoprzecinkowy, *m* określa łączną liczbę cyfr znaczących w liczbie, zaś *n* liczbę cyfr po przecinku – typ ten jest rekomendowany do przechowywania wartości walutowych;
- date – data z dokładnością do 1 dnia.

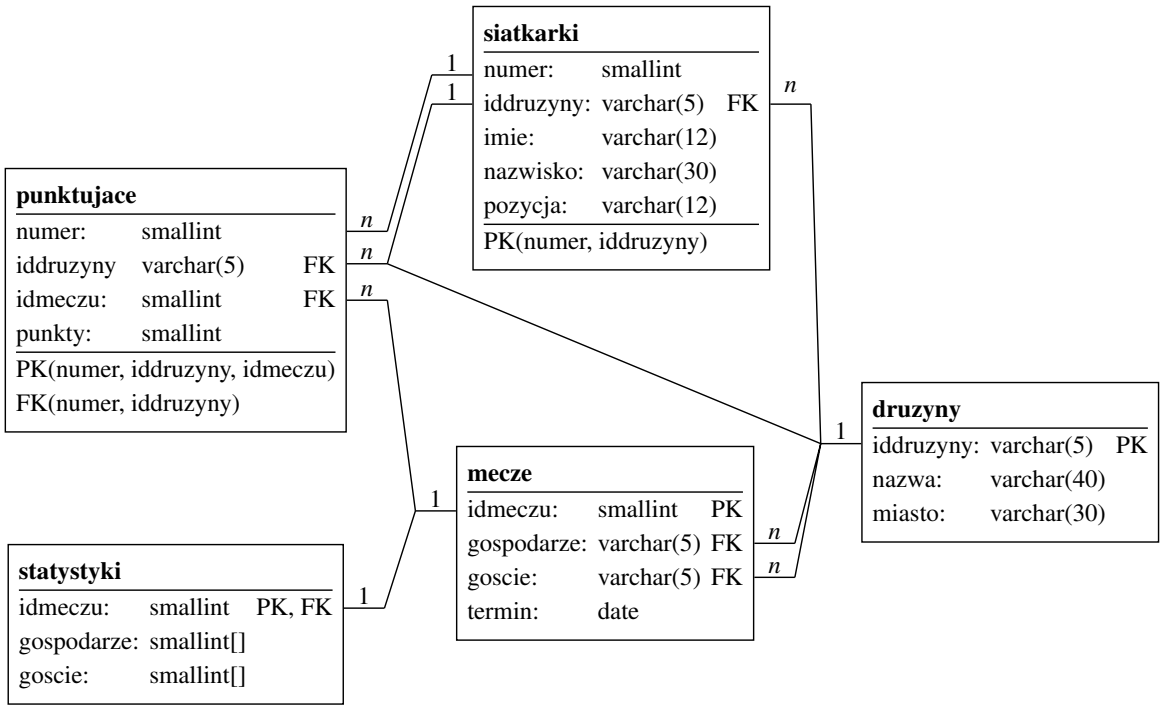
Klucze - definicje pojęć

- Zbiór identyfikujący – dowolny atrybut lub zestaw atrybutów pozwalających na jednoznaczną identyfikację krotki w relacji.
- Klucz – każdy minimalny zbiór identyfikujący, tzn. taki, że żaden jego właściwy podzbiór nie jest wystarczający do jednoznacznej identyfikacji krotki; klucz stanowi więc możliwie najprostszy zestaw atrybutów wystarczający do rozróżnienia wszystkich krotek w relacji,
- Klucz prosty – klucz złożony z pojedynczego atrybutu,
- Klucz złożony – klucz składający się z więcej niż jednego atrybutu,
- Klucz podstawowy (główny) – wybrany klucz preferowany przez użytkownika,
- Klucz obcy – atrybut lub ich zestaw występujący w danej relacji, a będący kluczem określonym w innej relacji, stosowany w celu identyfikacji elementów dla ewentualnego złączenia,
- Atrybut kluczowy – atrybut wchodzący w skład przynajmniej jednego klucza.

Klucze – uwagi

- Jeżeli żaden z atrybutów występujących w schemacie relacji nie pozwala na jednoznaczną identyfikację krotek, to zwykle wprowadza się (sztucznie) nowy atrybut jednoznacznie identyfikujący każdą krotkę, np.: numer PESEL, numer NIP, sygnatura książki w bibliotece, numer rejestracyjny samochodu itp.
- Klucz jest określany dla schematu relacji; co jest kluczem a co nie, nie zależy od aktualnej relacji (wartości zmiennej relacyjnej). Pojęcie klucza jest związane z istnieniem zależności funkcyjnych między atrybutami. Dla danego schematu relacji może istnieć wiele kluczy.
- W realnych bazach danych, nie ma w zasadzie przeszkód, aby pewne wiersze powtarzały się. Jeżeli przy definiowaniu schematu relacji nie zdefiniuje się żadnego klucza, ani nie zażąda się aby rekordy były indeksowane unikatowo, to wprowadzenie dwóch lub więcej identycznych wierszy do tabeli jest możliwe. Powtarzające się wiersze tabeli nazywa się duplikatami. Wiersze takie nie są rozróżnialne, a więc celowość reprezentowania więcej niż jednego takiego obiektu jest wątpliwa.

Baza danych *Liga siatkówki kobiet*



Algebra relacji

Algebra relacji zawiera proste, ale efektywne sposoby konstruowania nowych relacji na podstawie już istniejących. Konstruowane relacje mogą być odpowiedziami na pytania o dane przechowywane w bazie danych.

Klasy działań tradycyjnej algebry relacji

- Działania na zbiorach: suma, iloczyn, różnica – wykonywane w odniesieniu do relacji.
- Działania powodujące usuwanie części relacji: selekcja i rzutowanie.
- Działania łączące krotki dwóch relacji: iloczyn kartezjański i różnego rodzaju złączenia.
- Przemianowanie – zmiana schematu relacji w odniesieniu do nazw atrybutów lub samej relacji.

Współczesne SZBD nie używają bezpośrednio algebry relacji jako języka zapytań. Najczęściej stosowanym językiem zapytań jest obecnie SQL. Wiele instrukcji SQLa, to po prostu „składniowy cukier" dla wyrażeń algebry relacji.

Strukturalny język zapytań

SQL (Structured Query Language) jest językiem obsługi baz danych zaimplementowanym w systemach zarządzania bazami danych. Posiada on akceptację ANSI oraz standard ISO. W praktyce jest standardowym językiem zapytań dla relacyjnych baz danych.

Cechy języka SQL:

- Jest językiem wysokiego poziomu, opartym na słownictwie języka angielskiego; jego wyrażenia mają określoną strukturę.
- Jest językiem deklaratywnym (nieproceduralnym), zorientowanym na wynik (użytkownik definiuje co chce otrzymać, ale nie pisze jak).
- Nie posiada instrukcji sterujących wykonaniem programu.
- Nie dopuszcza rekurencji.
- Zawiera logikę trójwartościową.
- Umożliwia definiowanie struktur danych, wyszukiwanie danych, oraz operacje na danych.

SQL – komponenty i standardy

Komponenty języka SQL:

- DDL (Data Definition Language) – język definiowania struktur danych (`CREATE`),
- DQL (Data Query Language) – język definiowania zapytań dla wyszukiwania danych, (`SELECT`),
- DML (Data Manipulation Language – język operacji na danych (`SELECT, INSERT, UPDATE, DELETE`),
- Instrukcje sterowania danymi – np. kontrola uprawnień użytkowników (`GRANT, REVOKE`).

1986 – pierwsza publikacja standardu ANSI;

1987 – pierwsza publikacja standardu ISO;

1989 – zweryfikowany standard, znany powszechnie jako SQL-89 lub SQL-1;

1992 – standard SQL-92 (SQL-2), standard do dzisiaj stosowany w produktach komercyjnych;

1999 – standard SQL-99 (SQL-3);

2003 – SQL-2003 będący poprawionym SQL-3

PostgreSQL

- PostgreSQL jest obiektowo-relacyjnym systemem zarządzania bazą danych obsługującym język zapytań SQL.
- PostgreSQL zawiera wiele obiektowych rozszerzeń takich jak możliwość definiowania nowych typów podstawowych, nowych operatorów i dziedziczenia typów tablic.
- System działa na niemal wszystkich platformach UNIX, systemach opartych na UNIX-ie (FreeBSD, GNU/Linux), systemach rodziny Windows.
- PostgreSQL jest darmowy i oferuje otwarty dostęp do kodu źródłowego (Open Source, licencja PostgreSQL – licencja podobna do BSD).
- PostgreSQL zawiera niemal wszystkie funkcje, które można odnaleźć w innych bazach danych, zarówno komercyjnych, jak i Open Source, a także kilka funkcji dodatkowych, właściwych tylko dla niego. PostgreSQL jest niemal niezawodny, posiada dobre wskaźniki wydajności, jest bardziej kompatybilny ze standardem SQL niż MySQL.
- Baza danych PostgreSQL może być wykorzystywana z poziomu wielu języków programowania, włączając w to: C, C++, Perl, Python, Java, Tcl oraz PHP.

`http://www.postgresql.org`

PostgreSQL 9.1.9

Architektura klient-serwer

Proces serwera (postgres) odpowiada za zarządzanie plikami bazy danych, akceptowanie połączeń i wykonywanie operacji na bazie w imieniu klienta. Dla każdego nowego połączenia tworzony jest na podstawie procesu głównego (master server process) oddzielny proces serwera. Proces główny jest zawsze aktywny.
Przykładowe aplikacje klienckie: phppgadmin, psql

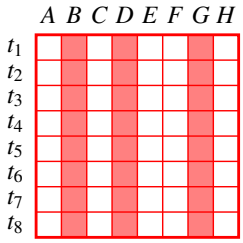
Instalacja/użytkowanie – Linux Debian/Mint/Ubuntu itp.

```
# z poziomu użytkownika root
apt-get install postgresql-9.1 phppgadmin
service postgresql start|stop|status
su postgres
createuser -P marcin
createdb cukiernia -O marcin

# dostęp do bazy danych -- z poziomu zwykłego użytkownika
psql cukiernia
http://localhost/phppgadmin
```


Rzutowanie (projekcja)

Rzutowanie (projekcja) oznacza ograniczenie informacji zawartej w relacji do wartości wybranych atrybutów.

$\pi_{B,D,G}(R)$



Z punktu widzenia algebry relacji, wynik rzutowania jest relacją więc nie zawiera duplikatów. Oznacza to, że relacja $\pi_{B,D,G}(R)$ może zawierać mniej krotek niż relacja R.

SQL

```
select kolumna1, kolumna2,..., kolumnaN from tabela;
select distinct kolumna1, kolumna2,..., kolumnaN from tabela;
```

Instrukcja select nie usuwa duplikatów, więc wynik jej działania nie zawsze jest relacją z teoretycznego punktu widzenia. Aby wymusić usuwanie duplikatów stosuje się dyrektywę distinct.

Rzutowanie w języku SQL – przykłady

```
select miasto, nazwa from druzyny;

select iddruzyny, nazwa, miasto from druzyny;

select * from druzyny;

select iddruzyny as identyfikator, nazwa, miasto from druzyny;

select iddruzyny as "identyfikator drużyny", nazwa, miasto
from druzyny;

select imie, nazwisko from siatkarki;

select imie, nazwisko from siatkarki limit 10;

select imie, nazwisko from siatkarki limit 5 offset 5;

select imie from siatkarki;

select distinct imie from siatkarki;
```

Klauzula limit ogranicza liczbę rekordów w wyniku.
Klauzula offset wskazuje ile *pierwszych* wyników ma zostać pominiętych.

Sortowanie wyników

Sortowanie wyników realizowane jest przez klauzulę order by, która powinna być umieszczana na końcu instrukcji select. Domyślnym sortowaniem jest sortowanie rosnące (opcja asc – ascending). Sortowanie malejące uzyskuje się w wyniku użycia opcji desc – descending. Sortowanie może odbywać się po kilku kolumnach, dla których możemy podać niezależny sposób sortowania. Zamiast nazwy kolumny można podać jej numer porządkowy

```
select pozycja, nazwisko, imie, numer from siatkarki
order by pozycja;

select pozycja, nazwisko, imie, numer from siatkarki
order by pozycja asc;

select pozycja, nazwisko, imie, numer from siatkarki
order by pozycja, nazwisko, imie;

select pozycja, nazwisko, imie, numer from siatkarki
order by pozycja desc, nazwisko, imie;
```

Do klauzuli order by można dodać nulls {first | last}, określając w jaki sposób mają być traktowane przy sortowaniu wartości null. Domyślnym jest nulls last przy sortowaniu rosnącym i nulls first przy malejącym.

Null

Wartości null to tzw. wartości puste, reprezentują one brak wartości danego atrybutu (chwilowy lub permanentny), który może być interpretowany jako:

- brak znajomości wartości danego atrybutu (dane niedostępne, wartość nieznana; np. data urodzenia, numeru NIP czy PESEL),
- niemożliwość wyznaczenia wartości danego atrybutu w krotce (atrybut niestosowalny, np. pojemność skokowa dla samochodów elektrycznych),
- niepewność co do istnienia tej wartości (np. gdy nie wiadomo, czy osoba posiada telefon).

Wartości null są różne od spacji, zera czy też pustego łańcucha znaków. W pewnych okolicznościach, wartości null są traktowane jak każde inne (np. wyświetlanie, porządkowanie) w innych okolicznościach nie podlegają przetwarzaniu (dają nieokreślone wartości funkcji).

Dwie wartości null **nie są** traktowane jako równe. Porównanie wartości null z dowolną wartością daje wartość logiczną unknown lub null (różną od true i false).

Duplikaty

Występowanie duplikatów powoduje pewne określone konsekwencje:

- Obiekty opisywane takimi krotkami nie są rozróżnialne, a więc celowość reprezentowania więcej niż jednego takiego obiektu jest wątpliwa.
- Usuwając informację o takim obiekcie z bazy należy wyraźnie sprecyzować, czy chodzi nam o usunięcie jednego jej wystąpienia, dwóch, trzech, itd., czy też wszystkich.
- Mogą pojawić się trudności związane z przechowywaniem takich tabel, jeżeli dostęp do ich elementów (wierszy) realizowany jest za pomocą indeksu unikatowego.
- W wyniku realizacji pewnych operacji na takich tabelach (np. złożenia lub iloczynu kartezjańskiego) następuje niekontrolowane i nie zawsze celowe powielanie informacji.
- Niektóre operacje na tabelach (np. łączenie) wymagają, aby tzw. tabela nadrzędna była indeksowana jednoznacznie względem pól łączących (lub aby pola te definiowały klucz) i bez spełnienia tego warunku operacje te nie mogą być realizowane; dlatego w większości przypadków, już na etapie projektowania schematów relacji bazy danych, definiuje się dla każdego schematu klucz główny.

Select distinct

Stosując klauzulę distinct on () można wskazać na podstawie jakich elementów (kolumny, wyrażenie) krotki będą traktowane jako duplikaty. W wyniku zostanie umieszczony pierwszy element z każdej z grup. O tym która to będzie krotka decyduje sposób sortowania. Zawartość nawiasów musi się pojawić jako skrajnie lewy element klauzuli order by.

```
select distinct imie, nazwisko from siatkarki;

select distinct on (imie) imie, nazwisko from siatkarki;

select distinct on (imie) imie, nazwisko
from siatkarki order by imie;

select distinct on (imie) imie, nazwisko
from siatkarki order by imie, numer;

select distinct on (imie, numer) imie, nazwisko
from siatkarki order by imie, numer;
```


Tablice

PostgreSQL posiada zdolność zapisywania tablic w tabelach. Nie jest to standardowa funkcja SQL. Aby w tabeli zadeklarować kolumnę jako tablicę, należy dodać [] po nazwie typu. Nie trzeba podawać liczby elementów (ale można choć ma to tylko wartość komentarza).
Indeksy tablic zaczynają się od 1. Element tablicy nie może być `null` (tablica tak).

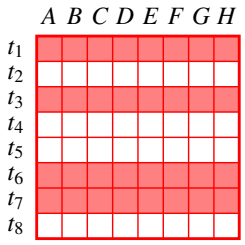
```
siatkowka=# select * from statystyki;
 idmeczu |    gospodarze    |      goscie
---------+------------------+------------------
       1 | {28,23,13,12}    | {26,25,25,25}
       2 | {25,25,25}       | {12,17,20}
       3 | {25,25,25}       | {10,15,17}
...

select idmeczu, gospodarze[1], goscie[1] from statystyki;
select idmeczu, gospodarze[2], goscie[2] from statystyki;
select idmeczu, gospodarze[1:2], goscie[1:2] from statystyki;
select idmeczu, gospodarze[4:5], goscie[4:5] from statystyki;
```

Tablice mogą mieć więcej wymiarów, np. squares integer[3][3].

Selekcja

Selekcja oznacza wybranie z relacji tych krotek, które spełniają zadany warunek (predykat Φ).

$$\sigma_\Phi(R)$$



Relacje R i $\sigma_\Phi(R)$ mają taki sam schemat.

SQL

```
select * from tabela where warunek;
```

Warunek Φ możemy budować z:

- stałych (w tym liczb i łańcuchów znakowych),
- atrybutów $A, B, \ldots$ pełniących w formule Φ rolę zmiennych wolnych,
- funkcji wbudowanych w systemie,
- symboli relacji wbudowanych w systemie (np. $<$, $>=$ itp.),
- spójników logicznych dopuszczalnych w systemie,
- nawiasów definiujących kolejność operacji.

Selekcja – przykłady

```
select * from siatkarki where pozycja = 'libero';

select * from siatkarki where numer = 2;

select * from mecze where termin = '2010-01-23';

select * from siatkarki where not pozycja = 'libero';

select * from siatkarki
where pozycja = 'libero' and iddruzyny = 'musz';

select * from siatkarki
where pozycja = 'libero' or pozycja = 'atakująca';
```

a	b	a and b	a or b
true	true	true	true
true	false	false	true
true	null	null	true
false	false	false	false
false	null	false	null
null	null	null	null

a	not a
true	false
false	true
null	null

Operatory relacyjne

<, <=, <>, !=, >, >=, =, **between and**, **not between and**

```
a between x and y    ≡   a >= x and a <= y
a not between x and y    ≡   a < x or a > y
```

```
select * from siatkarki where imie > 'K' order by imie;

select * from statystyki
where gospodarze[5] > 15 or goscie[5] > 15;

select * from mecze
where termin between '2009-11-01' and '2009-11-15'
order by termin;

select * from siatkarki
where nazwisko between 'N' and 'R'
order by nazwisko desc;

select * from punktujace
where punkty not between 10 and 20
order by iddruzyny, numer, punkty;
```


Porównania z wartością null

Przedstawione operatory relacyjne zwracają **null**, jeżeli co najmniej jeden operand ma wartość **null**, np. `7 = null` daje w wyniku **null**.
Porównanie z wartością **null** realizujemy za pomocą konstrukcji:
`wyrażenie is null` lub `wyrażenie is not null`.

`wyrażenie1 is distinct from wyrażenie2`
Jeżeli wyrażenia są różne od null, to konstrukcja jest równoważna operatorowi <>.
Jeżeli oba są równe null zwracane jest false, jeżeli jedno – zwracane jest true.

`wyrażenie1 is not distinct from wyrażenie2`
Jeżeli wyrażenia są różne od null, to konstrukcja jest równoważna operatorowi =.
Jeżeli oba są równe null zwracane jest true, jeżeli jedno – zwracane jest false.

`wyrażenie is [not] true|false|unknown`
Wyrażenie musi być typu Boolean. Konstrukcje te traktują null jako trzecią wartość logiczną unknown (a nie brak wartości). Wynikiem jest zawsze true lub false.

Porównania z wartością null – przykłady

```
select * from statystyki where gospodarze[4] is null;

select * from statystyki where gospodarze[5] is not null;

select * from statystyki
where gospodarze[4] = (goscie[4] - 2);

select * from statystyki
where gospodarze[4] is not distinct from (goscie[4] - 2);

select * from statystyki
where gospodarze[5] != 15;

select * from statystyki
where gospodarze[5] is distinct from 15;
```

Operatory arytmetyczne

$+, -, *, /, \%, \hat{}$

Selekcja i rzutowanie

Selekcję i rzutowanie można połączyć w jednym zapytaniu. Wyznaczamy wówczas $\pi_L(\sigma_\Phi(R))$, gdzie Φ jest warunkiem selekcji, zaś L jest zbiorem atrybutów, na które rzutujemy wynik.

```
select pozycja, nazwisko, imie from siatkarki
where pozycja='libero' or pozycja='atakująca'
order by pozycja desc, nazwisko, imie;

select nazwisko, imie, pozycja, iddruzyny
from siatkarki
where (nazwisko > 'K' and nazwisko < 'M')
or (iddruzyny = 'stal')
or (iddruzyny = 'pila' and pozycja != 'libero')
order by nazwisko, imie;

select gospodarze, goscie, idmeczu from statystyki
where gospodarze[1] > goscie[1]
and gospodarze[2] > goscie[2]
and gospodarze[3] > goscie[3]
order by idmeczu desc;
```

Należenie do zbioru – in, not in

Operatory in i not in są odpowiednikami algebraicznych operatorów $\in$ i $\notin$.

```
select nazwisko, imie, pozycja
from siatkarki
where imie = 'Anna' or imie = 'Anita' or imie = 'Alicja'
order by nazwisko, imie;

select nazwisko, imie, pozycja
from siatkarki
where imie in ('Anna', 'Anita', 'Alicja')
order by nazwisko, imie;

select nazwisko, imie, pozycja
from siatkarki
where imie not in ('Anna', 'Anita', 'Alicja')
and imie < 'B'
order by nazwisko, imie;
```


Funkcje matematyczne

Nazwa funkcji	**Opis**
abs(x)	wartość bezwzględna
exp(x)	exponens
ln(x)	logarytm naturalny z x
log(b,x)	logarytm o podstawie b z x
div(x,y)	część całkowita dzielenia x przez y
mod(x,y)	reszta z dzielenia x przez y
pi()	π
pow(x,y), power(x,y)	podnosi x do potęgi y
random()	zwraca losową liczbę pomiędzy 0,0 a 1,0
round(x,s)	zaokrąglenie do s miejsc po przecinku
sqrt(x)	pierwiastek kwadratowy z x
trunc(x,d)	obcina liczbę x do d miejsc po przecinku
sin(x), cos(x), tan(x), cot(x)	funkcje trygonometryczne

SQL jako kalkutalor

```
siatkowka=# select exp(2);
       exp
------------------
 7.38905609893065
(1 row)

siatkowka=# select pi();
        pi
-------------------
 3.14159265358979
(1 row)

siatkowka=# select pow(2,10);
 pow
------
 1024
(1 row)

siatkowka=# select 2*(12-7);
 ?column?
----------
       10
(1 row)
```


Obliczenia w kwerendach

`||` – operator konkatenacji
`::` – rzutowanie na wskazany typ

```
select nazwisko || ' ' || imie as Siatkarki, pozycja
from siatkarki
where iddruzyny = 'musz'
order by 1;

select nazwisko || ' ' || substr(imie, 1, 1) || '.' as Siatkarki,
       pozycja
from siatkarki
where iddruzyny = 'musz'
order by 1;

select gospodarze[1] + gospodarze[2] + gospodarze[3]
       as "Suma punktów"
from statystyki order by 1 desc;

select round(gospodarze[1]::numeric/goscie[1],4)
from statystyki;
```

Null w obliczeniach

Null może zastąpić dowolną wartość. Nie należy do żadnego konkretnego typu.
Podstawową regułą matematyczną dla wartości null jest ich propagacja.
Działanie arytmetyczne zawierające null da w wyniku null.

nullif(x,y) – funkcja zwraca null, jeżeli pierwszy argument jest równy drugiemu, w pozostałych przypadkach zwraca wartość pierwszego argumentu.
coalesce(x,y,z,...) – funkcja zwraca pierwszą wartość z listy różną od null, wstawiając ją do najwyższego typu danych.

```
select gospodarze[1] + gospodarze[2] + gospodarze[3]
     + gospodarze[4] + gospodarze[5] as "Suma punktów"
from statystyki order by 1 desc;

select gospodarze[1] + gospodarze[2] + gospodarze[3]
     + coalesce(gospodarze[4], 0)
     + coalesce(gospodarze[5], 0) as "Suma punktów"
from statystyki order by 1 desc;
```


Wyrażenie warunkowe

```
case when condition then result
    [when ...]
    [else result]
end
```

Do obliczenia wartości wyrażenia case stosowana jest pierwsza klauzula when ze spełnionym warunkiem. Jeżeli nie użyto klauzuli else, to system wstawia domyślną klauzulę else null. Wyrażenia case można w sobie zagnieżdżać.

```
select idmeczu, gospodarze, goscie as "goście",
(case when (gospodarze[1] > goscie[1]) then 1 else 0 end +
 case when (gospodarze[2] > goscie[2]) then 1 else 0 end +
 case when (gospodarze[3] > goscie[3]) then 1 else 0 end +
 case when (gospodarze[4] > goscie[4]) then 1 else 0 end +
 case when (gospodarze[5] > goscie[5]) then 1 else 0 end)
as "wygrane gospodarze",
(case when (gospodarze[1] < goscie[1]) then 1 else 0 end +
 case when (gospodarze[2] < goscie[2]) then 1 else 0 end +
 case when (gospodarze[3] < goscie[3]) then 1 else 0 end +
 case when (gospodarze[4] < goscie[4]) then 1 else 0 end +
 case when (gospodarze[5] < goscie[5]) then 1 else 0 end)
as "wygrane goście"
from statystyki;
```

Działania na typach związanych z datą i czasem

```
<data/czas> - <data/czas> = <interwał>
<data/czas> + <interwał>  = <data/czas>
<interwał> * (/) <liczba>  = <interwał>
```

```
select date '2001-09-28' + integer '7';
  -- date '2001-10-05'
select date '2001-09-28' + time '03:00';
  -- timestamp '2001-09-28 03:00:00'
select interval '1 day' + interval '1 hour';
  -- interval '1 day 01:00:00'
select timestamp '2001-09-28 01:00' + interval '23 hours';
  -- timestamp '2001-09-29 00:00:00'
select time '01:00' + interval '3 hours';
  -- time '04:00:00'
select date '2001-10-01' - date '2001-09-28';
  -- integer '3' (days)
select date '2001-09-28' - interval '1 hour';
  -- timestamp '2001-09-27 23:00:00'
select time '05:00' - time '03:00';
  -- interval '02:00:00'
select 900 * interval '1 second';
  -- interval '00:15:00'
```


Funkcje operujące na typach związanych z datą i czasem

Nazwa SQL	**Opis**
age(timestamp[, timestamp])	wiek
current_date	bieżąca data
current_time	bieżący czas
date_part(text, timestamp)	część daty (czasu)
now()	bieżąca data i czas
extract(field from timestamp)	część daty (czasu)

Dostępne pola dla funkcji extract (wybrane): century, day, decade, dow (day of week, 0-6), doy (day of year), hour, millennium, milliseconds, minute, month, quarter, second, week, year. W przypadku funkcji date_part nazwy te należy umieścić w apostrofach.

Operator overlaps pozwala sprawdzić, czy dwa okresy nachodzą (np. częściowo) na siebie:

```
(start1, end1) overlaps (start2, end2)
(start1, length1) overlaps (start2, length2)
```

Funkcje operujące na typach związanych z datą i czasem – przykłady

```
select age(timestamp '1997-01-17');

select current_date;

select current_time;

select termin, gospodarze, goscie
from mecze
where extract(month from termin) = 12
order by termin;

select termin, gospodarze, goscie
from mecze
where date_part('month', termin) = 12
order by termin;

select termin, gospodarze, goscie
from mecze
where termin >= current_date
order by termin;
```


Funkcje operujące na łańcuchach znaków

Nazwa SQL	**Opis**
lower(string)	konwersja na małe litery
position(substring in string)	pozycja w łańcuchu znaków
trim([leading | trailing | both] [characters] from string)	usunięcie wskazanego znaku
upper(string)	konwersja na wielkie litery
length(string)	długość tekstu
substr(string, from [, count])	wycięcie fragmentu tekstu

```
select lower(nazwa) from druzyny;

select substr(nazwa, 1, 5), miasto from druzyny;

select nazwa, miasto
from druzyny
where length(miasto) < 10 and substr(miasto, 1, 1) = 'B';
```

Porównywanie wzorców – like

```
string like pattern [escape escape_character]
string not like pattern [escape escape_character]
```

- Definiując wzorzec jako znaki specjalne używane są: % – dowolny ciąg znaków, _ – pojedynczy znak.
- Like porównuje wzorzec z całym łańcuchem znaków. Jeżeli chcemy wyszukać jakiś fragment w środku napisu, należy na początku i na końcu wzorca umieścić znak %.
- Opcjonalna część wyrażenia pozwala zdefiniować symbol ucieczki do wprowadzania znaków specjalnych. Domyślnie jest to ukośnik wsteczny.
- PostgreSQL dostarcza również konstrukcję ilike (nie należy do standardu SQL), działającą analogicznie jak like, ale nie rozróżniającą wielkości znaków.
- PostgreSQL dostarcza również operatory (nie należą do standardu SQL): ~~ (like), ~~* (ilike), !~~ (not like) i !~~* (not ilike).

Porównywanie wzorców – like – przykłady

Wyrażenie	Wartość
'abc' like 'abc'	true
'abc' like 'a%'	true
'abc' like '_b_'	true
'abc' like 'c'	false

```
select imie, nazwisko
from siatkarki
where nazwisko like 'Sz%'
order by nazwisko;

select imie, nazwisko
from siatkarki
where nazwisko ilike '%sz%' or nazwisko ilike '%cz%'
order by nazwisko;

select nazwa, miasto
from druzyny
where miasto like '_% %_';
```

Porównywanie wzorców – similar to

```
string similar to pattern [escape escape_character]
string not similar to pattern [escape escape_character]
```

Operator similar to zachowuje się podobnie jak like, ale używa standardowej definicji wyrażeń regularnych z języka SQL. Wspiera dodatkowe meta symbole do definiowania wzorców:

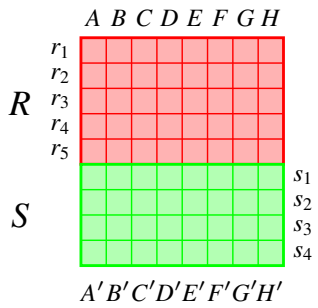
- `|` – alternatywa,
- `*` – powtórzenie 0 lub więcej razy poprzedniego elementu,
- `+` – powtórzenie 1 lub więcej razy poprzedniego elementu,
- `()` – grupowanie elementów.
- `[[: :]]` – specyfikacja klasy znaków (jak w POSIX) – dostępne nazwy klas to: *alnum, alpha, blank, cntrl, digit, graph, lower, print, punct, space, upper, xdigit.*

Porównywanie wzorców – similar to – przykłady

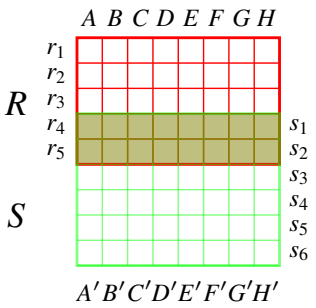
Wyrażenie	Wartość
'abc' similar to 'abc'	true
'abc' similar to 'a'	false
'abc' similar to '%(b|d)%'	true
'abc' similar to '(b|c)%'	false

```
select nazwa, miasto from druzyny
where miasto similar to '(B|W)%' -- '[BW]%'
order by miasto;

select imie, nazwisko from siatkarki
where nazwisko similar to '%(Sz|sz|Cz|cz)%' -- '%[SsCc]z%'
order by nazwisko;

select iddruzyny, nazwa  from druzyny
where iddruzyny similar to '%b+%';

select iddruzyny, nazwa from druzyny
where iddruzyny similar to '%bb+%';
```

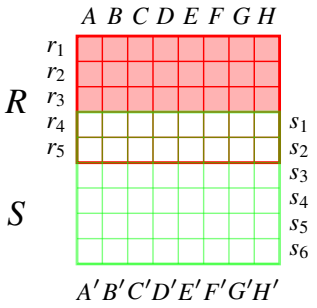
Łączenie zapytań



$R \cup S$
union



$R \cap S$
intersect



$R \setminus S$
except

Operacje sumy, przecięcia (iloczynu) i różnicy odpowiadają odpowiednim operacjom algebry zbiorów. Są one wykonywane na relacjach (zbiorach krotek). Relacje będące operandami muszą mieć ten sam schemat z dokładnością do dziedzin atrybutów.

SQL

```
select ...
union | intersect | except
select ... ;
```


Union – przykłady

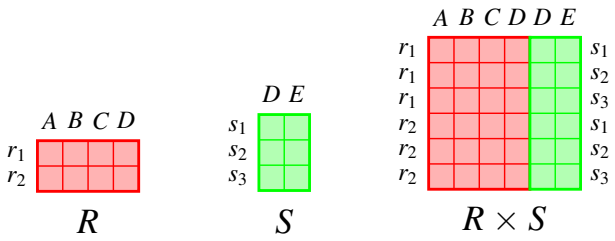
```
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 1
union
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 10
union
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 12;

select distinct iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu in (1, 10, 12);

select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 1
union
select iddruzyny, numer from siatkarki
where iddruzyny = 'musz' and pozycja = 'rozgrywająca'
order by 2;
```

Użycie operatora union (również intersect i except) powoduje usunięcie duplikatów z wyniku. Operatory te mają wersję z przyrostkiem all (np. union all), której użycie powoduje zaniechanie usuwania duplikatów.

Składanie zapytań – przykłady

```
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 1
intersect
select iddruzyny, numer from siatkarki
where iddruzyny = 'musz' and pozycja = 'atakująca'
order by 2;

select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 1
intersect
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 10
except
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 12
union
select iddruzyny, numer from punktujace
where iddruzyny = 'musz' and idmeczu = 19
order by 2;
```


Iloczyn kartezjański



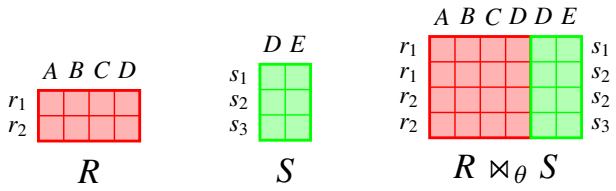
Iloczyn kartezjański definiuje się dla dowolnych relacji.

SQL

```
select * from tabela1, tabela2;
```

```
select * from druzyny, siatkarki;

select nazwa, nazwisko, imie from druzyny, siatkarki;

select a.iddruzyny as gospodarze, b.iddruzyny as goscie
from druzyny a, druzyny b
where a.iddruzyny <> b.iddruzyny;

select nazwa, nazwisko, termin from druzyny, siatkarki, mecze;
```

Złączenie θ



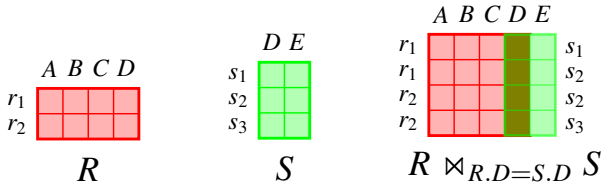
Złączenie θ jest dwuargumentowym działaniem na relacjach. Relacja wynikowa $R \bowtie_\theta S$ zawiera krotki powstałe z połączenia pary krotek (pierwsza z relacji R druga z S), które spełniają warunek logiczny θ.
Jeżeli warunek θ jest zawsze prawdziwy, to $R \bowtie_\theta S = R \times S$.

SQL

```
select ... from tabela1 join tabela2 on warunek;
```

```
select nazwa, miasto, nazwisko, imie, numer
from druzyny join siatkarki on nazwisko > 'K' and miasto < 'M';

select nazwa, miasto, nazwisko, imie, numer
from druzyny join siatkarki
on nazwisko like '%k%' or nazwa like '%k%';
```


Równozłączenie



Równozłączenie jest szczególnym przypadkiem θ-złączenia, w którym warunek θ dla relacji R, S ma postać $R.A = S.A \wedge R.B = S.B \wedge \ldots$, gdzie $A, B, \ldots$ są wspólnymi (niekoniecznie wszystkimi) atrybutami schematów obu relacji. W schemacie relacji wynikowej powtarzające się atrybuty są pomijane.

SQL

```
select ...
from tabela1 join tabela2 using(atrybut1,atrybut2,...);
```

```
select nazwisko, imie, nazwa
from siatkarki join druzyny using(iddruzyny)
where pozycja = 'rozgrywająca';

select nazwisko, imie, punkty
from siatkarki join punktujace using(numer, iddruzyny);
```

Złączenie naturalne

Złączenie naturalne $R \bowtie S$ jest szczególnym przypadkiem równozłączenia – warunek równozłączenia dotyczy wszystkich wspólnych atrybutów.

SQL

```
select ... from tabela1 natural join tabela2;
```

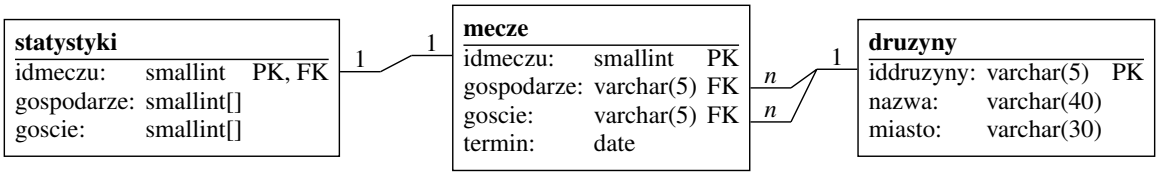
```
select nazwisko, imie, nazwa
from siatkarki natural join druzyny
where pozycja = 'rozgrywająca';

select nazwisko, imie, punkty
from siatkarki natural join punktujace;

-- brak powiązania w strukturze bazy danych
select numer, iddruzyny, punkty, gospodarze
from punktujace natural join statystyki;

-- brak wspólnych atrybutów => iloczyn kartezjański
select nazwa, termin from druzyny natural join mecze;

-- zapytanie niepoprawne
select idmeczu, termin from mecze natural join statystyki;
```


Łączenie wielu tabel

```
select nazwa, nazwisko, imie, punkty
from druzyny natural join siatkarki natural join punktujace;

select nazwisko, imie, termin, punkty
from siatkarki natural join punktujace natural join mecze
order by termin;

select nazwa, nazwisko, imie, punkty, termin
from druzyny natural join siatkarki
natural join punktujace natural join mecze
order by termin;

select kompozycje.nazwa, zamowienia.cena, klienci.miasto
from kompozycje join zamowienia using(idkompozycji)
join klienci on klienci.idklienta = zamowienia.idodbiorcy
where zamowienia.termin = '2010-03-27';

select nazwa, kompozycje.cena, termin
from kompozycje join zamowienia using (idkompozycji)
join zapotrzebowanie using (idkompozycji);
```

Aliasy

Możliwe jest użycie tej samej relacji wielokrotnie w zapytaniu, ale wymaga to zastosowania indywidualnych aliasów dla każdej *kopii*.




```sql
select gospodarze, goscie, termin from mecze;

select d1.nazwa, d2.nazwa, termin
from druzyny d1 join mecze on d1.iddruzyny = gospodarze
join druzyny d2 on d2.iddruzyny = goscie;

select d1.nazwa, d2.nazwa, s.gospodarze, s.goscie, termin
from druzyny d1 join mecze on d1.iddruzyny = gospodarze
join druzyny d2 on d2.iddruzyny = goscie
join statystyki s using(idmeczu);
```


Typy złączeń

Omawiane do tej pory złączenia, to złączenie wewnętrzne. Relacja wynikowa $R \bowtie_\theta S$ zawierająca tylko te konkatenacje krotek obu relacji wejściowych, dla których spełnione jest kryterium złączenia.

- Złączenie lewostronne zewnętrzne (left [outer] join) – wynik zawiera wszystkie krotki z R uzupełnione tymi krotkami z S, które spełniają warunek θ,
- Złączenie prawostronne zewnętrzne (right [outer] join) – wynik zawiera wszystkie krotki z S uzupełnione tymi krotkami z R, które spełniają warunek θ,
- Złączenie zewnętrzne pełne (full [outer] join), tzn. złączenie zawierające wszystkie krotki relacji R i S uzupełnione wartościami null, w przypadku gdy do danej nie pasuje żadna krotka z drugiej relacji
- Złączenie zewnętrzne typu union, tzn. złączenie zawierające wszystkie krotki relacji R nie pasujące do żadnej krotki relacji S uzupełnione tymi krotkami z relacji S, które nie pasują do żadnej krotki relacji R.

```sql
select imie, nazwisko, idmeczu
from siatkarki left join punktujace using(numer, iddruzyny);

select d.nazwa, s.nazwisko
from druzyny d [left|right|full] join siatkarki s
on substr(d.nazwa, 1, 1) = substr(s.nazwisko, 1, 1);
```

Funkcje agregujące

Nazwa funkcji	Opis
avg(x)	średnia arytmetyczna
count(*)	liczba wierszy spełniających określone warunki
count(x)	liczba wierszy, w których kolumna `x` ma wartość różną od `null`
max(x)	maksimum
min(x)	minimum
stddev(x)	odchylenie standardowe
sum(x)	suma
variance(x)	wariancja

W przypadku stosowania funkcji min i max do kolumn typu varchar ciągi porównywane są po uprzednim uzupełnieniu ich z prawej strony spacjami.
Powyższe funkcje ignorują wartość null.
Funkcje sum i avg przyjmują jako argument kolumnę o typie numerycznym.

Funkcje agregujące – przykłady

```
select count(*) from siatkarki;

select count(*) from siatkarki where iddruzyny = 'musz';

select count(*) from siatkarki natural join punktujace
where iddruzyny = 'musz' and idmeczu = 1;

select count(gospodarze[5]) from statystyki;

select avg(punkty) from punktujace;

select max(punkty) from punktujace;

select min(punkty), max(punkty) from punktujace;

select sum(punkty) from punktujace natural join siatkarki
where imie = 'Katarzyna' and nazwisko = 'Gajgał';

select min(gospodarze[1]) as "I set", min(gospodarze[2]) as "II set",
min(gospodarze[3]) as "III set", min(gospodarze[4]) as "IV set",
min(gospodarze[5]) as "V set"
from statystyki;
```

Funkcje agregujące – uwagi

Funkcja agregująca nie może wystąpić w klauzuli where. Klauzula where decyduje o tym, które krotki zostaną użyte do obliczeń z funkcją agregującą i jest obliczana przed obliczaniem wartości funkcji agregującej.

NIEPOPRAWNIE:

```
select imie, nazwisko, punkty
from siatkarki natural join punktujace
where punkty > avg(punkty);
```

POPRAWNIE:

```
select imie, nazwisko, punkty
from siatkarki natural join punktujace
where punkty > (select avg(punkty) from punktujace);
```

Podzapytanie stanowi niezależne zadanie obliczeniowe i tylko ono zawiera funkcję agregującą. Obliczenia z funkcją agregującą są niezależne od obliczeń w głównym zapytaniu.

Klauzula Group by

- Klauzula group by służy do grupowania (podziału na rozłączne podzbiory) krotek będących wynikiem selekcji (wynik działania klauzuli where).
- Klauzula group by musi wystąpić bezpośrednio po klauzulach from lub where.
- Grupowanie może dotyczyć więcej niż jednego atrybutu.
- Zastosowanie klauzuli group by oznacza, że wybierane atrybuty mogą wystąpić wyłącznie jako argumenty funkcji agregujących lub jako element grupowania.

```
select nazwisko, imie, avg(punkty), count(distinct punkty)
from punktujace natural join siatkarki
where iddruzyny = 'musz'
group by nazwisko, imie;
```

UWAGA: W powyższym przykładzie nie można zmienić kolejności klauzul where i group by.
Chociaż nie można porównywać ze sobą dwóch wartości null, to w kontekście klauzul group by, order by i distinct wartość null jest traktowana jako identyczna z inną wartością null lub jako jej duplikat.

Grupowanie – przykłady

```
select iddruzyny, count(*) from siatkarki
group by iddruzyny;

select iddruzyny, count(*) from siatkarki
group by iddruzyny, nazwisko
order by iddruzyny;

select nazwisko, imie, sum(punkty), nazwa
from siatkarki natural join druzyny natural join punktujace
group by nazwisko, imie, nazwa
order by nazwa, sum(punkty) desc;

select nazwisko, imie, min(punkty), max(punkty), nazwa
from siatkarki natural join druzyny natural join punktujace
group by nazwisko, imie, nazwa
order by nazwa, sum(punkty) desc;

select m.gospodarze, sum(s.gospodarze[1] + s.gospodarze[2]
+ s.gospodarze[3] + coalesce(s.gospodarze[4], 0)
+ coalesce(s.gospodarze[5], 0)) as "Suma punktów"
from statystyki s join mecze m using(idmeczu)
group by m.gospodarze
order by 2 desc;
```


Podzapytania skalarne

- Podzapytanie skalarne jest zwykłą instrukcją select umieszczoną w nawiasie okrągłym, która zwraca co najwyżej jedną krotkę zawierającą tylko jedną wartość.
- Podzapytanie skalarne może być częścią wyrażenia – zwracana przez nie wartość jest używana do wyznaczenia wartości takiego wyrażenia.
- Jeżeli podzapytanie skalarne nie zwraca żadnej krotki, to jako jego wynik przyjmowana jest wartość null.
- Podzapytanie skalarne może odwoływać się do nazw (zmiennych) z otaczającego je zapytania, które są traktowane jak stałe przy każdej ewaluacji podzapytania.
  Podzapytanie odwołujące się do atrybutów z zapytania nadrzędnego nazywamy skorelowanym.

```
select nazwisko, imie,
  (select max(punkty) from punktujace
   where punktujace.numer = siatkarki.numer
   and punktujace.iddruzyny = siatkarki.iddruzyny) as maksimum
from siatkarki order by 3 desc;
```

Podzapytania skalarne – przykłady

```
select nazwisko, imie, punkty,
  (select max(punkty)
   from punktujace
   where punktujace.numer = siatkarki.numer
   and punktujace.iddruzyny = siatkarki.iddruzyny) as maksimum
from siatkarki natural join punktujace order by 1;
```

```
select nazwa,
 (((select sum(statystyki.gospodarze[1])
    from statystyki join mecze using(idmeczu)
    where mecze.gospodarze = druzyny.iddruzyny)
 + (select sum(statystyki.goscie[1])
    from statystyki join mecze using(idmeczu)
    where mecze.goscie = druzyny.iddruzyny))::numeric(5,2) /
 ( (select count(statystyki.gospodarze[1])
    from statystyki join mecze using(idmeczu)
    where mecze.gospodarze = druzyny.iddruzyny)
 + (select count(statystyki.goscie[1])
    from statystyki join mecze using(idmeczu)
    where mecze.goscie = druzyny.iddruzyny)))::numeric(5,2)
from druzyny;
```


Klauzula having

Klauzula where służy do selekcji rekordów, które następnie poddawane są grupowaniu i używane są do obliczeń. Klauzula having służy do selekcji grup rekordów po tym jak zostaną one pogrupowane i wyznaczone są wartości funkcji agregujących.

W klauzuli having można agregować dowolne atrybuty relacji wymienionych w klauzuli from, ale bez agregacji w klauzuli having mogą wystąpić tylko te atrybuty, które wymieniono w klauzuli group by.

```
select nazwisko, imie, sum(punkty), nazwa
from siatkarki natural join druzyny natural join punktujace
group by nazwisko, imie, nazwa
having sum(punkty) > 200
order by sum(punkty) desc;
```

```
select nazwisko, imie, sum(punkty)
from siatkarki natural join punktujace
group by nazwisko, imie
having count(*) > 10
order by sum(punkty) desc;
```

Dodawanie rekordów – insert into

Instrukcja insert into jest jedyną metodą wstawiania nowych krotek do relacji (w języku SQL). W praktyce wiele narzędzi oferuje własne metody kopiowania dużej ilości danych.

SQL

```
insert into tabela values(lista wartości);
insert into tabela(lista atrybutów) values(lista wartości);
```

```
insert into druzyny values('pila', 'PTPS Piła', 'Piła');

insert into mecze values
  (1, 'organ', 'musz', '2009-10-25'),
  (2, 'alubb', 'stal', '2009-10-24'),
  (3, 'pila', 'gedan', '2009-10-24');
```

Lista wartości zawiera wartości dla kolejnych atrybutów oddzielone przecinkami. Należy podać je w takiej samej kolejności, w jakiej występują kolumny w tabeli. Napisy należy umieszczać w apostrofach (’). Umieszczenie apostrofu w ciągu znaków wprowadzanym do bazy danych jest możliwe po poprzedzeniu go znakiem \.

Dodawanie rekordów – insert into z listą atrybutów

- W rozszerzonej wersji instrukcji insert into lista wartości musi pasować do podanej listy atrybutów. Kolejność atrybutów może być dowolnie ustalona.
- Atrybuty ze schematu relacji, które pominięto na liście atrybutów, otrzymują wartość domyślną (jeżeli ją zdefiniowano), albo wartość null. Jeżeli wstawienie żadnej z tych wartości nie jest możliwe, to krotka nie zostanie dodana do relacji.
- Jako „wartości” można użyć słowa kluczowe null i default (użycie wartości domyślnej). W przypadku wstawiania danych do atrybutu typu serial można użyć albo słowo kluczowe default, albo pominąć nazwę atrybutu na liście.

```
insert into siatkarki(iddruzyny, numer, pozycja)
values('musz', 3, 'przyjmująca');

insert into siatkarki(iddruzyny, numer, imie, nazwisko)
values('musz', 4, null, null);

insert into mecze values(default, 'organ', 'musz', default);

insert into mecze(gospodarze, goscie, termin)
values('pila', 'gedan', default);
```

Wstawianie rekordów – podzapytania

UWAGA: Podzapytanie musi zwracać pojedynczą wartość. W drugim przypadku nie można użyć jednego podzapytania, które zwróci od razu dwie szukane wartości.

```
insert into mecze values(
default,
(select iddruzyny from druzyny where miasto = 'Bielsko-Biała'),
(select iddruzyny from druzyny where miasto = 'Łódź'),
default);

insert into punktujace values(
(select numer from siatkarki where nazwisko = 'Bełcik'),
(select iddruzyny from siatkarki where nazwisko = 'Bełcik'),
1, 12);

insert into punktujace
select numer, iddruzyny, 1, 12 from siatkarki
where nazwisko = 'Bełcik';
```


Tworzenie tabel z użyciem instrukcji select

Użycie instrukcji select into powoduje:

- Utworzenie tabeli o wskazanej nazwie;
- Użycie nazw (ewentualnie aliasów) wybranych atrybutów jako nazw atrybutów tworzonej tabeli;
- Użycie typów wybranych atrybutów jako typów atrybutów tworzonej tabeli.

```
select numer, iddruzyny, imie, nazwisko
into atakujace
from siatkarki
where pozycja = 'atakująca';

select a.iddruzyny as gospodarze, b.iddruzyny as goscie
into zestawienie
from druzyny a, druzyny b
where a.iddruzyny <> b.iddruzyny;
```

Usuwanie danych – delete

W instrukcji delete nie można używać nazw skorelowanych do relacji użytej w klauzuli delete from.

Usuwanie wszystkich danych z tabeli wykonuje się za pomocą instrukcji truncate. Jest ona bardziej wydajna przy dużych tabelach niż delete, ale nie jest tworzona kopia zapasowa i przy użyciu jej wewnątrz transakcji, nie można cofnąć zmian w oparciu o rollback.

SQL

```
delete from tabela [where warunek];
truncate tabela;
```

```
delete from punktujace where punkty < 3;

delete from punktujace
where punkty < (select avg(punkty) from punktujace);

truncate punktujace;
```


Aktualizacja danych – update

SQL

```
update tabela set atrybut1 = wartość1 [,atrybut2 = wartość2,...]
[where warunek];

update tabela set (atrybut1, atrybut2,...)
= (wartość1, wartość2,...) [where warunek];
```

Wszystkie przypisania w klauzuli set są realizowane równolegle. W ostatnim przykładzie nąstapi zamiana wartości atrybutów a i b w relacji tab.

```
update mecze set termin = termin + 1
where idmeczu between 6 and 10;

update mecze
set gospodarze =
  (select iddruzyny from druzyny where miasto = 'Bydgoszcz'),
goscie = (select iddruzyny from druzyny where miasto = 'Łódź'),
termin = '2009-10-28' where idmeczu = 1;

update tab set a = b, b = a;
```