



# Relacyjne Bazy Danych

Andrzej M. Borzyszkowski  
PJATK/ Gdańsk

materiały dostępne elektronicznie  
<http://szuflandia.pjwstk.edu.pl/~amb>

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski



## Perspektywy

- Bardziej skomplikowane zapytanie może zostać zapamiętane  
**CREATE VIEW towar\_zysk AS**  
**SELECT \*, cena - koszt AS zysk FROM towar**
  - zapamiętuje pytanie
  - późniejsze użycie odnosi się do treści tabeli z momentu tego użycia**SELECT \* FROM towar\_zysk**
  - jest zawsze równoważne zapytaniu  
**SELECT \*, cena - koszt AS zysk FROM towar**

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski



# Język SQL, cz. 2, operowanie na danych (*data manipulation language*) (uzupełnienia)

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski



## Perspektywy a tabele tymczasowe

- Perspektywa jest czym innym niż tabela tymczasowa  
**CREATE TEMP TABLE towar\_zysk (**  
**nr int PRIMARY KEY,**  
**opis varchar(64) , koszt numeric(7,2) ,**  
**cena numeric(7,2) , zysk numeric(7,2) )**  
**INSERT INTO towar\_zysk**  
**SELECT \*, cena - koszt AS zysk FROM towar**
  - wstawi do utworzonej wcześniej tabeli wynik obliczenia operacji SELECT
  - po zmianie zawartości tabeli towarów pytania  
**SELECT \*, cena - koszt AS zysk FROM towar**  
**SELECT \* FROM towar\_zysk**
    - dadzą *różne* odpowiedzi, nową i starą wartość zysku

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski



## Perspektywy c.d.

- Tabela tymczasowa może być używana tak jak każda tabela, w szczególności można do niej wstawiać i z niej usuwać krotki
- Operacje UPDATE i DELETE dla perspektyw nie są oczywiste
  - DELETE from towar\_zysk**  
**WHERE zysk/koszt<0.05**
    - można sobie wyobrazić realizację powyższego polecenia jako  
**DELETE from towar**  
**WHERE (cena-koszt)/koszt<0.05**
    - ale jak miałyby działać poniższa operacja na tabeli towar?  
**UPDATE towar\_zysk**  
**SET zysk=zysk\*1.1**

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

5

## Instrukcja SELECT – wyrażenia warunkowe w klauzuli WHERE

- Pojedyncze wartości: **WHERE cena > 3.14**
- Relacja pomiędzy wartością a zbiorem wartości:  
**WHERE miasto NOT IN ('Gdańsk', 'Gdynia', 'Sopot')**  
**WHERE koszt >= ALL ( SELECT koszt FROM towar )**
- Istnienie elementów: **WHERE NOT EXISTS ( SELECT \* FROM towar WHERE towar.koszt >= 3.14 )**
- Jednoznaczność elementów:  
**SELECT \* FROM zamowienie**  
**WHERE NOT**  
**klient\_nr MATCH UNIQUE ( SELECT nr FROM klient )**  
( to się nie powinno zdarzyć, jeśli nr jest kluczem w tabeli klientów )

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

7

## Perspektywy 3.

- PostgreSQL do wersji 9.2 nie przewidywał operacji UPDATE i DELETE dla perspektyw
  - amb=> create view test as select \* from towar;**  
**CREATE VIEW**  
**amb=> delete from test where nr=6;**  
**ERROR: cannot delete from view "test"**  
**PODPowiedź: You need an unconditional ON DELETE DO INSTEAD rule or an INSTEAD OF DELETE trigger.**
    - od wersji 9.3 dla szczególnie prostych perspektyw, definiowanych w oparciu o pojedynczą tabelę, bez grupowania, można używać operacje INSERT, DELETE i UPDATE
    - od wersji 9.4 perspektywa można dopuszczać pewne kolumny bez możliwości aktualizacji podczas gdy inne z możliwością
    - inne systemy zarządzania bazami danych mają/mogą mieć pewne możliwości operowania na perspektywach

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

6

## Instrukcja SELECT – złączenie naturalne

- SELECT opis, kod**  
**FROM towar NATURAL JOIN kod\_kreskowy K (kod,nr)**
- alias dla tabeli jednocześnie wprowadził aliasy dla kolejnych atrybutów tabeli
  - NATURAL JOIN nie wymaga podania warunku złączenia, tabele złączane są w/g pasujących nazw atrybutów
  - nawet jeśli zbieżność jest przypadkowa
- SELECT nr, nazwisko, opis, data\_wyslki**  
**FROM (klient NATURAL JOIN towar) NATURAL JOIN zamowienie**

nr	nazwisko	opis	data_wyslki
1	Kuśmerek	układanka drewniana	17.03.2021
2	Chodkiewicz	układanka typu puzzle	22.01.2021
3	Szczęsna	kostka Rubika	9.02.2021
4	Łukowski	Linux CD	9.03.2021

© Andrzej M. Borzyszkowski

Bazy Danych

8

## Instrukcja SELECT – złączenie naturalne, c.d.

- Wynik naturalnego złączenia jest prawidłowy *jedynie* przy założeniu, że odpowiadające sobie atrybuty mają identyczne nazwy w różnych tabelach
  - założenie mało prawdopodobne w dużej bazie danych

<i>nr</i>	<i>nazwisko</i>	<i>imie</i>	<i>miasto</i>
1	Kuśmierek	Małgorzata	Gdynia
2	Chodkiewicz	Jan	Gdynia
3	Szczęсна	Jadwiga	Gdynia
4	Łukowski	Bernard	Gdynia

<i>nr</i>	<i>klient_nr</i>	<i>data_wysylki</i>	<i>nr</i>	<i>opis</i>	<i>cena</i>
1	3	17.03.2021	1	układanka drewniana	21.95
2	8	22.01.2021	2	układanka typu puzzle	19.99
3	15	9.02.2021	3	kostka Rubika	11.49
4	13	9.03.2021	4	Linux CD	2.49

© Andrzej M. Borzyszkowski

Bazy Danych

9

## Instrukcja SELECT – zagnieżdżenie

- Podaj dane klientów którzy złożyli zamówienia po 1 marca 2020  
**SELECT nazwisko FROM klient**  
**WHERE nr IN ( SELECT klient\_nr**  
**FROM zamowienie**  
**WHERE data\_zlozenia > '2020-3-1'**  
**)**
  - zagnieżdżona tabela użyta w warunku służy jako zbiór
  - możemy najpierw wykonać wewnętrzne zapytanie, a potem zewnętrzne
  - *brak* korelacji, w zapytaniu podrzędnym nie ma odwołania do wiersza z tabeli zewnętrznej

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

11

## Instrukcja SELECT – theta-złączenie, konieczność aliasów

- Czasami wygodne może być stosowanie aliasów dla nazw tabel  
**SELECT K.nr, nazwisko, imie, data\_zlozenia**  
**FROM klient K, zamowienie Z WHERE K.nr = klient\_nr**
- Ale czasami jest niezbędne:
  - podaj nazwiska par klientów z tego samego miasta  
**SELECT I.nazwisko, II.nazwisko, I.miasto**  
**FROM klient I, klient II**  
**WHERE I.miasto = II.miasto**  
**AND I.nr < II.nr**
  - konieczna zmiana nazwy tabel
  - *nie* jest to złączenie względem pary klucz obcy–klucz główny
  - użycie innego warunku nazywa się  $\Theta$  (theta)-złączeniem
  - dodatkowy warunek ma trochę uporządkować wydruk

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

10

## Zagnieżdżenia: korelacja

- Podaj dane klientów, których nazwiska się powtarzają:  
**SELECT imie, nazwisko, miasto**  
**FROM klient**  
**WHERE nazwisko IN (**  
**SELECT nazwisko**  
**FROM klient**  
**GROUP BY nazwisko HAVING count (nazwisko) > 1**  
**)**
  - *brak* korelacji, w zapytaniu podrzędnym nie ma odwołania do wiersza z tabeli zewnętrznej
  - mimo, że ta sama tabela przeglądana jest dwukrotnie, brak korelacji powoduje brak potrzeby zmiany nazwy
  - wiadomo w każdym miejscu o czyje nazwisko chodzi

© Andrzej M. Borzyszkowski

Bazy Danych

12

## Zagnieżdżenia: korelacja c.d.

- Podaj dane klientów, którzy złożyli zamówienie po 1 marca 2020

```
SELECT nazwisko
FROM klient
WHERE EXISTS (
  SELECT *
  FROM zamowienie
  WHERE klient.nr = klient_nr AND data_zlozenia > '2020-3-1'
)
```

- wewnętrzny SELECT odwołuje się do tabeli zewnętrznej
- nie możemy wykonać wewnętrznego zapytania w oderwaniu od reszty
- występuje korelacja
- moilibyśmy użyć aliasu dla tabeli zewnętrznej, ale konieczności nie ma

13

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Zagnieżdżenia: korelacja 3.

- Podaj dane klientów, których nazwiska się powtarzają:

```
SELECT imie, nazwisko, miasto
FROM klient K
WHERE EXISTS (
  SELECT *
  FROM klient
  WHERE nazwisko=K.nazwisko AND nr != K.nr
)
```

- występuje korelacja, wewnętrzne pytanie zawiera odwołanie do wiersza z tabeli zewnętrznej
- dotatkowo, tutaj tabela przeglądana w podrzędnym zapytaniu jest ta sama co w zewnętrznym, występuje konieczność nazwania zewnętrznej tabeli

14

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Zagnieżdżenia: wydajność

- Zagnieżdżenie skorelowane wymaga wykonania innego podzapytania w każdym wierszu
  - czyli złożoność wykonania będzie radykalnie większa
  - zagnieżdżenie skorelowane

```
EXPLAIN SELECT * FROM zamowienie
WHERE ( SELECT miasto FROM klient
        WHERE nr = klient_nr ) = 'Gdańsk'
```

```
QUERY PLAN Seq Scan on zamowienie (cost=0.00..11128.00 rows=7 width=30)
Filter: (((SubPlan 1)::text = 'Gdańsk'::text)
```

- zagnieżdżenie nieskorelowane

```
EXPLAIN SELECT * FROM zamowienie
WHERE klient_nr IN ( SELECT nr FROM klient
                    WHERE miasto = 'Gdańsk' )
```

```
QUERY PLAN Hash Join (cost=12.02..40.80 rows=680 width=30)
Hash Cond: (zamowienie.klient_nr = klient.nr)
```

15

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Zagnieżdżenia: wydajność c.d.

- Zagnieżdżenie skorelowane nominalnie wymaga wykonania innego podzapytania w każdym wierszu
  - ale optymalizator zapytań może znaleźć inny plan wykonania
  - zagnieżdżenie skorelowane:

```
EXPLAIN SELECT * FROM zamowienie
WHERE EXISTS ( SELECT * FROM klient
               WHERE nr = klient_nr and miasto = 'Gdańsk')
```

```
QUERY PLAN Hash Join (cost=12.02..40.80 rows=680 width=30)
Hash Cond: (z.klient_nr = k.nr)
```

- zagnieżdżenie nieskorelowane

```
EXPLAIN SELECT * FROM zamowienie
WHERE klient_nr IN ( SELECT nr FROM klient
                    WHERE miasto = 'Gdańsk' )
```

```
QUERY PLAN Hash Join (cost=12.02..40.80 rows=680 width=30)
Hash Cond: (zamowienie.klient_nr = klient.nr)
```

16

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Zagnieżdżenia: wydajność 3.

- Optymalizator zapytań może znaleźć nawet lepszy plan wykonania
  - zagnieżdżenie skorelowane:

```
EXPLAIN SELECT imie, nazwisko, miasto FROM klient K
WHERE EXISTS ( SELECT * FROM klient
WHERE nazwisko=K.nazwisko AND nr != K.nr )
```

```
QUERY PLAN Hash Semi Join (cost=13.60..26.34 rows=1 width=214)
Hash Cond: ((k.nazwisko)::text = (klient.nazwisko)::text)
```

- zagnieżdżenie nieskorelowane:

```
EXPLAIN SELECT imie, nazwisko, miasto FROM klient
WHERE nazwisko IN ( SELECT nazwisko FROM klient
GROUP BY nazwisko HAVING count (nazwisko) > 1 )
```

```
QUERY PLAN Hash Semi Join (cost=18.00..31.80 rows=160 width=214)
Hash Cond: ((public.klient.nazwisko)::text =
(public.klient.nazwisko)::text)
```

17

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Zagnieżdżenie w atrybucie wynikowym

- EXPLAIN SELECT towar\_nr, sum(ilosc) AS razem  
FROM pozycja  
GROUP BY towar\_nr

```
QUERY PLAN HashAggregate (cost=40.60..42.60 rows=200 width=12)
Group Key: towar_nr
```

-> Seq Scan on pozycja (cost=0.00..30.40 rows=2040 width=8)

- EXPLAIN SELECT nr, ( SELECT sum(ilosc) AS razem  
FROM pozycja WHERE towar\_nr=towar.nr  
FROM towar

```
QUERY PLAN Seq Scan on towar (cost=0.00..13291.50 rows=390 width=12)
SubPlan 1
```

-> Aggregate (cost=34.04..34.05 rows=1 width=8)

-> Bitmap Heap Scan on pozycja (cost=23.45..34.01 rows=10 width=4)

Recheck Cond: (towar\_nr = towar.nr)

-> Bitmap Index Scan on pozycja\_pk (cost=0.00..23.45 rows=10 w...)

18

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Instrukcja SELECT – brak kwantyfikatora ogólnego w języku SQL

- Podaj nazwiska klientów, którzy zamówili każdy towar w dostępny ofercie:

```
SELECT nr, imie, nazwisko
FROM klient K
WHERE NOT EXISTS -- nie istnieje
( SELECT *
FROM towar T
WHERE NOT EXISTS -- towar niezamawiany
( SELECT *
FROM zamowienie Z INNER JOIN pozycja ON
Z.nr=zamowienie_nr AND
K.nr = klient_nr AND T.nr = towar_nr
))
```

- SQL nie ma konstrukcji dla kwantyfikatora ogólnego FORALL, konieczne podwójne przeczenie dla EXISTS

19

© Andrzej M. Borzyszkowski

Bazy Danych

## Instrukcja SELECT – kwantyfikator ogólny, inne rozwiązanie

- Podaj nazwiska klientów, którzy zamówili każdy towar w dostępny ofercie:

```
SELECT nr, imie, nazwisko
FROM klient K
WHERE NOT EXISTS -- nie istnieje
( ( SELECT nr FROM towar )-- wszystkie towary
EXCEPT -- oprócz
( SELECT towar_nr -- towary zamawiane przez tego klienta
FROM zamowienie Z INNER JOIN pozycja ON
Z.nr = zamowienie_nr AND K.nr = klient_nr
))
```

- użycie operatora teoriomnościowego
  - nie każdy system baz danych implementuje EXCEPT

20

© Andrzej M. Borzyszkowski

Bazy Danych

## Instrukcja SELECT – kwantyfikatory ogólny, teoretyczne rozwiązanie

- Podaj nazwiska klientów, którzy zamówili każdy towar w dostępny ofercie:

```
SELECT nr, imie, nazwisko
FROM klient K WHERE
(( SELECT towar_nr          -- towary zamawiane
  FROM zamowienie Z INNER JOIN pozycja ON
    Z.nr=zamowienie_nr AND K.nr = klient_nr
  )
CONTAINS
( SELECT nr FROM towar )    -- wszystkie towary
)
```

- SQL nie pozwala na porównanie tabel
- oryginalny system R firmy IBM posiadał implementację predykatu zawierania

Bazy Danych © Andrzej M. Borzyszkowski

21

## Przykład „brak kwantyfikatora ogólnego w języku SQL” – wydajność

- Podwójne przeczenie dla EXISTS:

```
Nested Loop Anti Join (cost=0.00..2051005.92 rows=85 width=136)
Join Filter: (NOT (SubPlan 1))
-> Seq Scan on klient k (cost=0.00..11.70 rows=170 width=136)
-> Materialize (cost=0.00..15.85 rows=390 width=4)
   -> Seq Scan on towar t (cost=0.00..13.90 rows=390 width=4)
      SubPlan 1
```

Bazy Danych © Andrzej M. Borzyszkowski

23

## Instrukcja SELECT – brak kwantyfikatora ogólnego w języku SQL, c.d.

- Podaj nazwiska klientów, którzy zamówili każdy towar w dostępny ofercie:

```
SELECT nr, imie, nazwisko
FROM klient K WHERE
( SELECT count (DISTINCT towar_nr)  -- towary zamawiane
  FROM zamowienie Z INNER JOIN pozycja ON
    Z.nr=zamowienie_nr AND K.nr = klient_nr
  )
=
( SELECT count(nr) FROM towar        -- wszystkie towary
  )
```

- korzystamy z tego, że każdy towar z tabeli pozycji musi występować w tabeli towarów (integralność referencyjna)
- stąd porównanie liczebności gwarantuje zawieranie

Bazy Danych © Andrzej M. Borzyszkowski

22

## Przykład „brak kwantyfikatora ogólnego w języku SQL” – wydajność

- Zagnieżdżenie skorelowane wersja 1 z EXCEPT:

```
Seq Scan on klient k (cost=0.00..47.55 rows=85 width=136)
Filter: (NOT (SubPlan 1))
SubPlan 1
-> HashSetOp Except (cost=0.00..82.25 rows=390 width=8)
   -> Append (cost=0.00..81.25 rows=400 width=8)
```

- Zagnieżdżenie skorelowane wersja 2 z porównaniem licznosci:

```
Seq Scan on klient k (cost=14.88..10803.08 rows=1 width=136)
Filter: ($0 = (SubPlan 2))
InitPlan 1 (returns $0)
-> Aggregate (cost=14.88..14.88 rows=1 width=8)
   -> Seq Scan on towar (cost=0.00..13.90 rows=390 width=4)
      SubPlan 2
```

Bazy Danych © Andrzej M. Borzyszkowski

24

## Instrukcja SELECT – zagnieżdżenia, warunek IN

- Podaj dane klientów, których imiona i nazwiska się powtarzają:  
**SELECT imie, nazwisko, miasto**  
**FROM klient**  
**WHERE ( imie, nazwisko ) IN (**  
    **SELECT imie, nazwisko**  
    **FROM klient**  
    **GROUP BY imie, nazwisko HAVING count ( nazwisko ) > 1**  
    **)**
- warunek należenia do zbioru stosowany jest do par wartości
- tabela zwracana w zapytaniu podrzędnym ma tyle kolumn, ile wartości w klauzuli WHERE

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

26

## Instrukcja SELECT – porównania z wartościami zagregowanymi c.d.

- Podaj dane o towarach o koszcie maksymalnym:  
**SELECT \* FROM towar**  
**WHERE koszt = (**  
    **SELECT max( koszt ) FROM towar**  
    **)**
- tabela 1x1, czyli pojedyncza wartość i porównanie z tą wartością
- Inne rozwiązanie  
**SELECT \* FROM towar**  
**WHERE koszt >= ALL (**  
    **SELECT koszt FROM towar**  
    **)**
- tabela o jednej kolumnie, czyli zbiór i porównanie z całym zbiorem

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

28

## Instrukcja SELECT – zagnieżdżenia, porównania z wartościami zagregowanymi

- Podaj dane o towarach o koszcie powyżej przeciętnej:  
**SELECT \***  
**FROM towar**  
**WHERE koszt > (**  
    **SELECT avg( koszt ) FROM towar**  
    **)**
- brak korelacji, nie ma konieczności zmiany nazwy
- tabela wynikowa z zapytaniu podrzędnym (1x1) jest traktowana jak pojedyncza wartość

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

27

## Porównania z wartościami zagregowanymi – wydajność

- Porównanie z pojedynczą wartością  
**EXPLAIN SELECT \* FROM towar**  
**WHERE koszt = (**  
    **SELECT max( koszt ) FROM towar**  
    **)**  
**Seq Scan on towar (cost=14.88..29.76 rows=2 width=178)**  
**Filter: (koszt = \$0)**
- Porównanie z całym zbiorem  
**EXPLAIN SELECT \* FROM towar**  
**WHERE koszt >= ALL (**  
    **SELECT koszt FROM towar**  
    **)**  
**Seq Scan on towar (cost=0.00..3295.75 rows=195 width=178)**  
**Filter: (SubPlan 1)**

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

29