Bazy Danych

Andrzej M. Borzyszkowski

materiały dostępne elektronicznie
https://inf.ug.edu.pl/~amb

Język SQL, cz. 2, operowanie na danych (*data manipulation language*) c.d.

© Andrzej M. Borzyszkowski
Bazy Danych

Cztery główne operacje / słowa kluczowe

- **INSERT** – realizuje aktualizację/wstawianie danych
- **SELECT** – główna operacja wyszukiwania danych
- **SELECT** [**ALL** | **DISTINCT**] *lista_atrybutów_wynikowych*
  [*lista_klauzul*];
  - *lista_atrybutów_wynikowych* rzut i zmiana nazwy kolumny
  - *lista_klauzul* obcięcie, złączenie, funkcje agregujące, porządkowanie
  - klauzule: **FROM** **WHERE** **ORDER BY** **GROUP BY** **HAVING**
- **UPDATE** – realizuje aktualizację/zmianę wartości danych
- **DELETE** – realizuje aktualizację/usuwanie danych

© Andrzej M. Borzyszkowski
Bazy Danych

Instrukcja SELECT – lista atrybutów

- Atrybut wynikowy jest albo gwiazdką * albo postaci
  wyrażenie_skalarne [[**AS**] *nazwa_kolumny*]
- * oznacza wszystkie atrybuty
  SELECT * FROM towar
  - wyświetla całą tabelę towarów
- *wyrażenie_skalarne* będzie najczęściej nazwą pojedynczego atrybutu
  SELECT imie, nazwisko FROM klient
- Realizuje rzut relacji: π[imie,nazwisko](Klient)
- **DISTINCT** usuwa powtarzające się wiersze w tabeli wynikowej, domyślnie jest **ALL**
  - cena usuwania nie jest błaha przy większych danych
  - niektóre implementacje porządkują wynik, nie jest to standard

© Andrzej M. Borzyszkowski
Bazy Danych

Instrukcja SELECT – atrybuty wynikowe

- *wyrażenie_skalarne* może odwoływać się do nazw atrybutów, ale zawierać dodatkowe obliczenia
- *nazwa_kolumny* będzie nazwą kolumny w tabeli wynikowej
- SELECT *, cena – koszt AS zysk FROM towar
  - dodaje nową kolumnę w wyświetlanym wyniku
  - zawiera ona wyniki obliczeń

nr	*opis*	*koszt*	*cena*	*zysk*
1	układanka drewniana	15,23	21,95	6,72
2	układanka typu puzzle	16,43	19,99	3,56
3	kostka Rubika	7,45	11,49	4,04
4	Linux CD	1,99	2,49	0,50
5	chusteczki higieniczne	2,11	3,99	1,88
6	ramka do fotografii 4'x6'	7,54	9,95	2,41

Instrukcja SELECT – atrybuty wynikowe c.d.

- Bardziej wymyślne wyrażenie

```
SELECT *, cena – koszt AS zysk,
    case when (cena-koszt)/koszt < 0 then 'ujemny'
       when (cena-koszt)/koszt < 0.4  then 'za mało'
       when cena is NULL then 'brak danych'
       else 'ok'
    end as opinia
FROM towar
```

nr	*opis*	*koszt*	*cena*	*zysk*	*opinia*
8	ramka do fotografii 3'x4'	13,36	19,95	6,59	ok
9	szczotka do zębów	0,75	1,45	0,70	ok
10	moneta srebrna z Papieżem	20,00	20,00	0,00	za mało
11	torba plastikowa	0,01	0,00	-0,01	ujemny
12	głośniki	19,73	25,32	5,59	za mało
13	nożyczki drewniane	8,18			brak danych
14	kompas wielofunkcyjny	22,10			brak danych

Instrukcja SELECT – atrybuty wynikowe c.d.

- Możliwość wykonania obliczeń wykracza poza proste operacje algebry relacji (rzut uogólniony)
- Dodatkowe obliczenia w wyrażeniu skalarnym nie muszą ograniczać się do atrybutów z tabel
- SELECT 2+2
- SELECT now()
- SELECT version()

```
                                   version
-------------------------------------------------------------------------------
 PostgreSQL 9.5.12 on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu
5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609, 64-bit
(1 row)
```

  - tabela wynikowa w ogóle nie odwołuje się do żadnej relacji

Instrukcja SELECT – różne warunki WHERE

- Podaj nazwiska klientów spoza Trójmiasta:

```
SELECT  nazwisko FROM klient
WHERE miasto NOT IN ('Gdańsk', 'Gdynia', 'Sopot')
```

  - warunek należenia do zbioru
- Podaj opis wszystkich ramek do fotografii, które mają podany wymiar w calach (tj. znak prim na końcu opisu)

```
SELECT opis FROM towar
WHERE opis LIKE 'ramka%' and opis LIKE E '%\''
```

  - dopasowanie wzorca tekstowego
- Wyświetl szczegóły zamówień złożonych w lutym 2019

```
SELECT * FROM zamowienie
WHERE data_zlozenia  BETWEEN '2019-02-01' AND '2019-02-29'
```

  - warunek dla zakresu dat

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 ***
- 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)

Instrukcja SELECT – klauzula ORDER BY

- Klauzula ORDER BY

  ORDER BY *lista_kolumn* [**DESC** | **ASC**]
- Występuje po klauzulach **FROM** i **WHERE**
- Wynikiem jest tabela, w której wiersze uporządkowano według atrybutów z listy kolumn, kolejność rosnąca (**ASC**, domyślnie) lub malejąca (**DESC**)

  SELECT * FROM towar ORDER BY koszt DESC
  - wyświetla tabelę towarów uporządkowaną według kosztów, zaczynając od największych

  SELECT * FROM towar ORDER BY koszt DESC LIMIT 3
  - dodatkowa opcja pozwalająca ograniczyć wyświetlanie

Instrukcja SELECT – funkcje agregujące

- *wyrażenie_skalarne* w części **SELECT** może być funkcją obliczaną dla wielu/wszystkich wierszy tabeli
  - jeśli nie wystąpi zmiana nazwy **AS** *nazwa_kolumny* to nazwa funkcji będzie nazwą w tabeli wynikowej

  SELECT count(*) FROM klient
  - zwraca liczbę klientów
  - tylko jedna kolumna, o nazwie „count”, i jeden wiersz
  - wynik może być użyty jako pojedyncza liczba

  SELECT count (DISTINCT nazwisko) FROM klient
  - usuwa powtórzenia przed podjęciem zliczania

  SELECT max(koszt), min(koszt), avg(koszt) AS średni
  FROM towar
  - wyświetla tabelę o jednym wierszu i trzech kolumnach

Instrukcja SELECT – klauzula GROUP BY

- Klauzula GROUP BY

  GROUP BY *lista_kolumn*
- Występuje po klauzulach **FROM** i **WHERE**
- Wynikiem jest tabela, w której zgrupowano wiersze o identycznych atrybutach z listy kolumn
- Elementy wyboru instrukcji **SELECT** mają obowiązek dawać jednoznaczną wartość dla każdej grupy:
  - albo muszą odwoływać się do atrybutów z listy kolumn, w/g których grupujemy
  - albo do funkcji agregujących

  SELECT towar_nr, count(zamowienie_nr), sum(ilosc)
  FROM pozycja
  GROUP BY towar_nr
  ORDER BY count(zamowienie_nr) DESC

Instrukcja SELECT – klauzula GROUP BY, c.d.

- Wymóg jednoznaczności dla wartości atrybutu traktowany jest w SQL formalnie
  - tzn. można odwoływać się do tylko atrybutów, w/g których następuje grupowanie
  - nie wystarczy gwarancja jednoznaczności poprzez użycie klucza kandydującego
  - w poniższym przykładzie trzeba dodać atrybut opis do grupowania, mimo że nie spowoduje to zmiany grup
    - SELECT towar.nr, opis, count(zamowienie_nr), sum(ilosc)
    - FROM pozycja INNER JOIN towar ON towar_nr=towar.nr
    - GROUP BY towar.nr, opis
    - ORDER BY count(zamowienie_nr) DESC
- W Postgresie wersji 9 w powyższym przykładzie można opuścić atrybut opis, grupowanie wg klucza gwarantuje jednoznaczność

Instrukcja SELECT – klauzula HAVING

- Klauzula HAVING

  HAVING *wyrażenie_warunkowe*
  - Występuje po innych klauzulach
  - Wynikiem jest tabela taka jak otrzymana poprzez użycie GROUP BY, ale dodatkowo z wyeliminowanymi grupami nie spełniającymi wyrażenia warunkowego
  - Brak GROUP BY oznacza, że cała tabela jest jedną grupą
  - Wyrażenie warunkowe odwołuje się do wartości, które można wyświetlić legalnie w SELECT

```
SELECT towar_nr, count(zamowienie_nr) FROM pozycja
GROUP BY towar_nr
HAVING count(zamowienie_nr) > 1
ORDER BY count(zamowienie_nr) DESC
```


Instrukcja SELECT – klauzula HAVING, użycie

- SELECT towar.nr, opis, count(zamowienie_nr)
  FROM pozycja INNER JOIN towar on towar_nr=towar.nr
  GROUP BY towar.nr, opis
  HAVING opis LIKE '%układanka%'
  - jest prawidłowe, ale nielogiczne i niesłuszne
  - HAVING jest słuszne, gdy odwołuje się do wartości zagregowanych
  - wartości pojedynczych krotek powinny być zbadane przed grupowaniem, w klauzuli WHERE

```
SELECT towar.nr, opis, count(zamowienie_nr)
FROM pozycja INNER JOIN towar on towar_nr=towar.nr
WHERE opis LIKE '%układanka%'
GROUP BY towar.nr, opis
```


Instrukcja SELECT – zagnieżdżenie

- Podaj nazwiska klientów, którzy założyli zamówienie po 1 marca 2019:

```
SELECT DISTINCT nazwisko
FROM klient K, zamowienie
WHERE K.nr = klient_nr AND data_zlozenia > '2019-3-1'
```

  - rozwiązanie to jest niezbyt szczęśliwe
  - jeśli dwóch występuje dwóch klientów o tym samym nazwisku, to tego nie zauważymy
  - użycie DISTINCT jest konieczne, ponieważ dla danego klienta może być wiele zamówień
  - właściwsze byłoby użycie SELECT DISTINCT nr, nazwisko
  - jeśli nie jesteśmy zainteresowani wyświetlaniem nr, to trzeba stosować grupowanie (DISTINCT GROUP BY)

Instrukcja SELECT – zagnieżdżenie, c.d.

- Właściwe rozwiązanie:

```
SELECT nazwisko FROM klient
WHERE nr IN ( SELECT klient_nr
              FROM zamowienie
              WHERE data_zlozenia > '2019-3-1'
             )
```

  - zagnieżdżona tabela użyta w warunku, tabela jednokolumnowa służy jako zbiór
  - nie jest obliczane złączenie
  - każdy klient jest wyświetlany co najwyżej raz (tzn. jeśli spełnia warunek)
  - jeśli powtarzają się nazwiska klientów spełniających warunek, to będą one uwzględnione

Instrukcja SELECT – zagnieżdżenie głębokie

- Podaj nazwiska klientów, którzy cokolwiek zamówili (tzn. złożyli niepuste zamówienie – puste też bywają)

```
SELECT nazwisko
FROM klient
WHERE nr IN (SELECT klient_nr
             FROM zamowienie
             WHERE nr IN (SELECT zamowienie_nr
                          FROM pozycja
                         )
             )
```

  - wielokrotne zagnieżdżenia, trzeba rozpatrywać od wewnątrz

Instrukcja SELECT – zagnieżdżenie w atrybucie wynikowym

- Podaj numery towarów wraz z ich całkowitymi wielkościami zamówień:

```
SELECT towar_nr, sum(ilosc) AS razem
FROM pozycja
GROUP BY towar_nr
```

- Podobne rozwiązanie:

```
SELECT nr, ( SELECT sum(ilosc) AS razem
             FROM pozycja
             WHERE towar_nr=towar.nr
           ) AS razem
FROM towar
```

  - zagnieżdżona tabela 1x1 użyta jako pojedyncza wartość
  - wyświetlone są wszystkie towary, nawet te niezamawiane

Instrukcja SELECT – zagnieżdżenie w klauzuli FROM, alias dla wyniku

- oblicz i zanalizuj zysk:

```
SELECT *,
    case when zysk/koszt < 0 then 'ujemny'
       when zysk/koszt < 0.4  then 'za mało'
       when cena is NULL then 'brak danych'
       else 'ok'
    end as opinia
FROM (SELECT *, cena – koszt AS zysk FROM towar) QQ
```

  - tabela w zagnieżdżeniu ma dodatkową kolumnę
  - tabela ta musi być nazwana i wówczas może być użyta jako źródło dla kolejnego wyszukiwania

Instrukcja SELECT – warunek niepustości

- Podaj nazwiska klientów, którzy założyli zamówienie po 1 marca 2019 – jeszcze jedno rozwiązanie:

```
SELECT nazwisko
FROM klient K
WHERE EXISTS (
  SELECT *
  FROM zamowienie
  WHERE K.nr = klient_nr AND data_zlozenia > '2019-3-1'
)
```

  - ponieważ testujemy tylko niepustość zbioru wierszy, więc nie zależy nam na szczegółowych wynikach
- wewnętrzny SELECT odwołuje się do tabeli zewnętrznej
  - jest to bardzo bliskie kwantyfikatora egzystencjalnego w rachunku krotek

Instrukcja SELECT – negatywne zapytanie

- Podaj nazwiska klientów, którzy założyli zamówienie po 1 marca 2019:

```
SELECT DISTINCT nazwisko
FROM klient K, zamowienie
WHERE K.nr = klient_nr AND data_zlozenia > '2019-3-1'
```

- Podaj nazwiska klientów, którzy nie założyli zamówienia po 1 marca 2019 ?
  - nie wiadomo, któremu warunkowi zaprzeczyć

```
data_zlozenia <= '2019-3-1'
```

  - oznacza zamówienie złożone wcześniej, ale jednak złożone
  - klient mógł złożyć zamówienia i przed i po podanej dacie

```
K.nr != klient_nr
```

  - jest totalnym nieporozumieniem, wyświetla klientów z cudzymi zamówieniami

Instrukcja SELECT – negatywne zapytanie 2

- Podaj nazwiska klientów, którzy nie założyli zamówienia po 1 marca 2019:

```
SELECT nazwisko FROM klient
WHERE nr NOT IN ( SELECT klient_nr FROM zamowienie
                  WHERE data_zlozenia > '2019-3-1' )
```

  - albo

```
SELECT nazwisko FROM klient K
WHERE NOT EXISTS (
  SELECT * FROM zamowienie
  WHERE K.nr = klient_nr AND data_zlozenia > '2019-3-1' )
```

- Będą jeszcze inne rozwiązania tego problemu

Instrukcja SELECT – operacje algebry relacji

- Podaj nazwiska klientów, którzy nie założyli zamówienia po 1 marca 2019 ?

```
SELECT nazwisko FROM klient
EXCEPT
SELECT nazwisko FROM klient
WHERE nr IN ( SELECT klient_nr FROM zamowienie
              WHERE data_zlozenia > '2019-3-1' )
```

  - operacja różnicy relacji
  - w tym przypadku rozwiązanie jest *nieprawidłowe*
  - może być dwóch klientów o tym samym nazwisku, jeden złożył zamówienie w badanym okresie, a drugi nie złożył
  - byłoby inaczej, gdyby wyświetlać nr klienta (wartość klucza)
- Istnieją też UNION oraz INTERSECT
  - w wersji z UNION ALL powtórzenia krotek są zachowane

Instrukcja UPDATE – składnia

- UPDATE *cel* SET *element* = *wartość*
  WHERE *warunek*
  - *cel* jest nazwą tabeli, w której aktualizujemy dane
  - *element* jest nazwą atrybutu, któremu przypisujemy *wartość*
  - klauzula WHERE wyznacza wiersze, w których będzie dokonana aktualizacja
  - ma ona identyczne znaczenie jak w instrukcji SELECT, w szczególności jej brak oznacza, że wszystkie wiersze będą aktualizowane
- SQL nie przewiduje możliwości aktualizacji kilku atrybutów w jednym poleceniu
  - niektóre implementacje dopuszczają taką możliwość

Instrukcja UPDATE – przykład

- UPDATE towar SET cena = 1.15
  WHERE nr=5
  - aktualizacja pojedynczego wiersza (klucz główny)
- UPDATE towar SET cena = cena*1.15
  WHERE opis LIKE '%układanka%'
  - aktualizacja wielu wierszy jednocześnie
- UPDATE towar SET cena = (
  SELECT cena FROM towar WHERE nr=5)
  - tabela 1x1 występuje w roli pojedynczej wartości (gdyby warunek WHERE w zagnieżdżonym zapytaniu nie odwoływał się do wartości kluczowej, polecenie UPDATE mogłoby produkować błąd)
  - brak warunku WHERE w poleceniu UPDATE oznacza, że jest globalne – dotyczy całej tabeli

Instrukcja DELETE

- DELETE FROM *cel*
  WHERE *warunek*
  - *cel* jest nazwą tabeli, z której usuwamy dane
  - klauzula WHERE wyznacza wiersze, w których będzie dokonana aktualizacja
  - ma ona identyczne znaczenie jak w instrukcji SELECT, w szczególności jej brak oznacza, że wszystkie wiersze są usuwane
- PostgreSQL i inne implementacje pozwalają na nieodwołalne usunięcie całej zawartości tabeli:
  TRUNCATE TABLE *cel*
- Uwaga: usuwanie wszystkich danych z tabeli, to nie jest to samo co usuwanie tabeli
  DROP TABLE *cel*

Instrukcja DELETE – przykład

- Usuń dane o klientach z Gdańska
  DELETE FROM klient
  WHERE miasto = 'Gdańsk'
- Usuń wszelkie informacje o zamówieniach składanych przez klientów z Gdyni

```
DELETE FROM zamowienie Z
WHERE ( SELECT miasto
        FROM klient K
        WHERE K.nr = Z.klient_nr
      ) = 'Gdynia'
```

  - klient_nr jest kluczem obcym w tabeli zamówień, jest więc dokładnie jeden klient dla tego zamówienia, wynikiem instrukcji SELECT jest tabela 1x1, czyli pojedyncza wartość