Bazy Danych

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytetu Gdańskiego

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

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

Cztery główne operacje / słowa kluczowe

- Stosowane są do tabel, nie zbiorów
  - wiersze mogą się powtarzać
  - kolejność wierszy gra rolę
- **SELECT** – główna operacja wyszukiwania danych,
  - realizuje zmianę nazwy, obcięcie, rzut i złączenie relacji
- **INSERT** – realizuje aktualizację/wstawianie danych
- **UPDATE** – realizuje aktualizację/zmianę wartości danych
- **DELETE** – realizuje aktualizację/usuwanie danych

- Notacja użyta dalej
  - [ ] oznacza element składniowy opcjonalny
  - | oznacza wybór jednego z elementów składniowych

Instrukcja INSERT – składnia

- **INSERT INTO *cel* [(*lista_elementów*)] *źródło;***
  - ***cel*** jest nazwą tabeli, do której wstawiamy dane
  - ***lista_elementów*** zawiera listę nazw atrybutów, którym chcemy nadać wartość, może być mniejsza niż pełna list tabeli ***cel***
  - ***źródło*** ma jedną z dwu postaci
    VALUES (*lista_wartości*)
    albo tabela otrzymana w wyniku operacji **SELECT**
- Od pewnej wersji PostgreSQL dopuszcza wygodniejszą formę wstawiania wielu wierszy:
  VALUES (*lista_wartości*) [,(*lista_wartości*)]*
  tzn. wymienienie wielu wierszy pod jednym słowem **VALUES**

Instrukcja INSERT – przykład

INSERT INTO kod_kreskowy
VALUES ('4892840112975', 17)

- wstawia jeden wiersz
- nadaje wartości atrybutom zadeklarowanym w definicji tabeli, w kolejności deklaracji
- nie można opuścić żadnego z atrybutów

INSERT INTO towar (opis, koszt)
VALUES ('donica duża', 26.43),
('donica mała', 13.36)

- wstawia dwa wiersze
- atrybuty „nr” oraz „cena” nie zostały wymienione
- będą miały wartość domyślną (kolejny numer / NULL)
- kolejność atrybutów nie musi być zgodna z kolejnością deklaracji w tabeli

Instrukcja INSERT – przykład, c.d.

- **INSERT INTO chwilowa**
  SELECT imie, nazwisko, ulica_dom
  FROM klient
  WHERE miasto = 'Gdańsk'
  - wstawia do utworzonej wcześniej tabeli 'chwilowa' całą tabelę otrzymaną w wyniku obliczenia operacji **SELECT**
- Celowe może być zdefiniowanie tabeli jako
  CREATE TEMP TABLE chwilowa (imię varchar(11), ……..
  - taka tabela jest usuwana po zakończeniu sesji
- **INSERT INTO towar (opis, koszt, cena)**
  VALUES ('ramka do fotografii 3''x4''', 13.36, NULL)
  - podwójny apostrof służy do wprowadzenia znaku apostrofu
  - można wprowadzić w jawny sposób wartość nieokreśloną

Instrukcja SELECT – składnia

- **SELECT [ALL | DISTINCT]** ***lista_atrybutów_wynikowych***
  [***lista_klauzul*** **];**
- ***lista_atrybutów_wynikowych*** realizuje m.in. rzut i zmianę nazwy kolumny
- ***lista_klauzul*** realizuje m.in. obcięcie i złączenie
- klauzule: **FROM WHERE ORDER BY GROUP BY HAVING**

SELECT DISTINCT imie, nazwisko
-- rzut na atrybuty
FROM klient
WHERE miasto = 'Gdańsk'
-- obcięcie do wierszy spełniających warunek

imię	*nazwisko*
Agnieszka	Kołak
Andrzej	Sosnowy
Barbara	Songin
Ewa	Hałasa
Jan	Soroczyński
Marzena	Niezabitowska-Nasiadko

Instrukcja SELECT – klauzula FROM

- Klauzula FROM
  FROM ***lista_tabel***
- Lista tabel nie może być pusta
- Wynikiem jest iloczyn kartezjański tabel
- **SELECT * FROM klient**
  - jedna tabela, iloczyn równy tej tabeli
- **SELECT * FROM towar, kod_kreskowy**
  - iloczyn kartezjański dwu tabel
- **SELECT * FROM klient, towar**
  - w obu tabelach występuje atrybut „nr”, czysto przypadkowa zbieżność
  - podając nazwę atrybutu, w przypadku takiej zbieżności, trzeba dodać nazwę tabeli

Instrukcja SELECT – klauzula WHERE

- Klauzula WHERE
  WHERE *wyrażenie_warunkowe*
- Występuje po klauzuli FROM
- Wynikiem jest wybór tych wierszy, które spełniają warunek
  SELECT * FROM klient WHERE miasto = 'Gdańsk'
  - obcięcie relacji w/g warunku **miasto = 'Gdańsk'**
  - $\sigma_{[miasto='Gdańsk']}$(Klient) (sigma)
- Warunek:
  - równość, nierówność itp. na atrybutach
  - należenie atrybutu do zbioru (tabela 1 kolumnowa)
  - operacje logiczne na prostszych warunkach
- Klauzula nie musi występować, wówczas wybrane są wszystkie wiersze tabeli

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 fotograf ii, które moją podany wymiar w calach (tj. znak prim na końcu opisu)
  SELECT opis FROM towar
  WHERE opis LIKE E'ramka%' and opis LIKE E'%\''
  - dopasowanie wzorca tekstowego
- Wyświetl szczegóły zamówień złożonych w marcu 2024
  SELECT * FROM zamowienie
  WHERE data_zlozenia BETWEEN '2024-03-01' AND '2024-03-31'
  - 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 – złączenie 1

- Złączenie jest wyborem pasujących wierszy w iloczynie kartezjańskim
  SELECT klient.nr, nazwisko, imie, data_zlozenia
  FROM klient, zamowienie WHERE klient.nr = klient_nr
  - bez warunku WHERE byłyby wszystkie pary wierszy
  - czyli iloczyn kartezjański
  - w obu tabelach występuje atrybut „nr”, trzeba wyjaśnić, o który chodzi
- Wygodne może być stosowanie *aliasów* dla nazw tabel
  SELECT K.nr, nazwisko, imie, data_zlozenia
  FROM klient K, zamowienie WHERE K.nr = klient_nr
  - w złączeniach wielokrotnie powtarzamy nazwę tabeli
  - ale jeśli alias jest zadeklarowany, musi być koniecznie używany

Złączenie, przykład

nr	nazwisko	imie
3	Szczęsna	Jadwiga
4	Łukowski	Bernard
5	Soroczyński	Jan
6	Niezabitowska-Nasiadko	Marzena
7	Kołak	Agnieszka
8	Kołak	Agnieszka

klient_nr	data_zlozenia
3	13.02.2024
3	23.02.2024
3	23.01.2024
4	22.02.2024
4	1.02.2024
5	4.02.2024
8	12.01.2024
8	7.01.2024

nr	nazwisko	imie	data_zlozenia
3	Szczęsna	Jadwiga	13.02.2024
3	Szczęsna	Jadwiga	23.02.2024
3	Szczęsna	Jadwiga	23.01.2024
4	Łukowski	Bernard	22.02.2024
4	Łukowski	Bernard	1.02.2024
5	Soroczyński	Jan	4.02.2024
8	Kołak	Agnieszka	12.01.2024
8	Kołak	Agnieszka	7.01.2024

Instrukcja SELECT – złączenie 2

- Inna składnia na złączenie

```
SELECT K.nr, nazwisko, imie, data_zlozenia
FROM klient K INNER JOIN zamowienie ON K.nr  =
  klient_nr
```

  - bezpośrednie odwołanie się do operacji złączenia w algebrze relacyjnej
  - deklaracja atrybutu klient_nr jako klucza obcego wskazującego na klient(nr) nie zwalnia z obowiązku napisania jawnego warunku dla złączenia
  - słowo kluczowe INNER jest domyślne (będą inne złączenia)

Instrukcja SELECT – przykłady podstawowe

- Podaj nazwiska i numery telefonów klientów z Gdyni

```
SELECT  nazwisko, telefon
FROM klient
WHERE miasto = 'Gdynia'
```

  - obcięcie i rzut w jednym
- Podaj opis i kod kreskowy wszystkich towarów

```
SELECT opis, kod
FROM towar T, kod_kreskowy
WHERE T.nr = towar_nr
```

  - złączenie

```
SELECT opis, kod
FROM towar T INNER JOIN kod_kreskowy
ON T.nr = towar_nr
```


SQL a rachunek krotek

- Zapytanie

```
SELECT atrybuty FROM tabele WHERE warunek
```

  - bezpośrednio przypomina konstrukcję

  $\{<t.A1,..,t.An> \mid r(t) \text{ AND } \Phi(t)\}$ lub $\{ t \mid r(t) \text{ AND } \Phi(t) \}$
- Zmienna *t*, która przebiega krotki, nie musi wystąpić w postaci jawnej
  - ale wygodnie jest myśleć, że wykonanie zapytania polega na pętli przebiegającej wszystkie krotki

```
SELECT T.opis, K.kod
FROM towar T, kod_kreskowy K
WHERE T.nr = K.towar_nr
```

  występują jawne nazwy dla krotek z tabel
  - nazwy atrybutów poprzedzone są nazwą (aliasem) tabeli

SQL a rachunek krotek, c.d.

- Zapytanie

  SELECT DISTINCT K.nazwisko
  FROM klient K, zamowienie Z
  WHERE K.nr = Z.klient_nr

  - oznacza

    { K.nazwisko | Klient(K) AND
    ∃ Z (Zamówienie(Z) AND K.nr=Z.klient_nr) }

  - w rachunku krotek występuje kwantyfikator egzystencjalny
  - w SQL jest on niejawny – rzut dotyczy istniejących par krotek, w szczególności istnieje zamówienie spełniające warunek
  - warunek Klient(K) od razu gwarantuje, że jest tylko skończona liczba krotek do rozpatrzenia

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

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

Instrukcja SELECT – atrybuty wynikowe c.d.

- Bardziej wymyślne wyrażenie

```
SELECT *, cena – koszt AS zysk,
case when (cena-koszt)/koszt < 0 then 'ujemn
  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 Papież	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 dany
14	kompas wielofunkcyjny	22,10			brak dany

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()**

```
              now
-------------------------------
 2024-03-11 21:21:09.451788+02
(1 row)
```

  - tabela wynikowa w ogóle nie odwołuje się do żadnej relacji

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 jednoznaczna 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 od wersji 9, można opuścić atrybut opis w powyższym przykładzie

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
```