Bazy Danych

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytetu Gdańskiego

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

Język SQL, operowanie na danych (*data manipulation language*) (uzupełnienia)

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
```


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
```

  - wstawia 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

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 miałaby działać poniższa operacja na tabeli towar?

```
UPDATE towar_zysk
SET zysk=zysk*1.1
```


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.
```


Perspektywy 4.

- Od Postgres wersji 9.3 dla szczególnie prostych perspektyw, definiowanych w oparciu o pojedynczą tabelę, bez grupowania, można używać operacji 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

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_wysylki
FROM (klient NATURAL JOIN towar) NATURAL JOIN
  zamowienie
```

nr	*nazwisko*	*opis*	*data_wysylki*
1	Kuśmierek	układanka drewniana	17.03.2024
2	Chodkiewicz	układanka typu puzzle	22.01.2024
3	Szczęsna	kostka Rubika	9.02.2024
4	Łukowski	Linux CD	9.03.2024

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

nr	nazwisko	imie	miasto
1	Kuśmierek	Małgorzata	Gdynia
2	Chodkiewicz	Jan	Gdynia
3	Szczęsna	Jadwiga	Gdynia
4	Łukowski	Bernard	Gdynia

nr	klient_nr	data_wysyll
1	3	17.03.2024
2	8	22.01.2024
3	15	9.02.2024
4	13	9.03.2024

nr	opis	cena
1	układanka drewniana	21.95
2	układanka typu puzzle	19.99
3	kostka Rubika	11.49
4	Linux CD	2.49

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 Θ (theta)-złączeniem
  - dodatkowy warunek ma trochę uporządkować wydruk

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..11455.00 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=12.14..39.89 rows=8 width=30)
  Hash Cond: (zamowienie.klient_nr = klient.nr)
```


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=12.14..39.89 rows=8 width=30)
  Hash Cond: (zamowienie.klient_nr = klient.nr)
```

  - zagnieżdżenie nieskorelowane

```
EXPLAIN SELECT * FROM zamowienie
WHERE klient_nr IN ( SELECT nr FROM klient
          WHERE miasto = 'Gdańsk' )
QUERY PLAN   Hash Join  (cost=12.14..39.89 rows=8 width=30)
  Hash Cond: (zamowienie.klient_nr = klient.nr)
```

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=13.82..26.75 rows=57 width=214)
  Hash Cond: ((k.nazwisko)::text = (klient.nazwisko)::text)
  Join Filter: (klient.nr < k.nr)
```

  - 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 Join  (cost=15.39..27.54 rows=170 width=214)
  Hash Cond: ((klient.nazwisko)::text = (klient_1.nazwisko)::text)
  ->  Seq Scan on klient  (cost=0.00..11.70 rows=170 width=214)
```


Zagnieżdżenie w atrybucie wynikowym

- **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 width=0)
```


Zagnieżdżenie w atrybucie wynikowym vs funkcja agregująca

- **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
```

  - ale wynik tego zapytania jest uboższy niż poprzedniego, wyświetlane są wyłącznie numery towarów zamawianych

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

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 tego klienta
          FROM zamowienie Z INNER JOIN pozycja ON
          Z.nr = zamowienie_nr AND K.nr = klient_nr
))
```

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

Instrukcja SELECT – kwantyfikator 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

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

Przykład „brak kwantyfikatora ogólnego w języku SQL” – wydajność

- Podwójne przeczenie dla EXISTS:

```
QUERY PLAN  Nested Loop Anti Join  (cost=0.00..2028854.90 rows=85
width=136)
  Join Filter: (NOT (SubPlan 1))
  ->  Seq Scan on klient k  (cost=0.00..11.70 rows=170 width=136)
```

- Zagnieżdżenie skorelowane wersja 1 z EXCEPT:

```
QUERY PLAN  Seq Scan on klient k  (cost=0.00..48.43 rows=85
width=136)
  Filter: (NOT (SubPlan 1))
  SubPlan 1
    ->  HashSetOp Except  (cost=0.00..82.25 rows=390 width=8)
```

- Zagnieżdżenie skorelowane wersja 2 z równością:

```
QUERY PLAN  Seq Scan on klient k  (cost=14.88..10835.56 rows=1
width=136)
  Filter: ((SubPlan 1) = $1)
  InitPlan 2 (returns $1)
    ->  Aggregate  (cost=14.88..14.88 rows=1 width=8)
```

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

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

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

Porównania z wartościami zagregowanymi – wydajność

- Porównanie z pojedynczą wartością

```
EXPLAIN SELECT * FROM towar
WHERE koszt = (
    SELECT max( koszt ) FROM towar
    )
```

```
QUERY PLAN  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
    )
```

```
QUERY PLAN  Seq Scan on towar  (cost=0.00..3295.75 rows=195
width=178)
  Filter: (SubPlan 1)
```