Relacyjne Bazy Danych

Andrzej M. Borzyszkowski
PJATK/ Gdańsk

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

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

Instrukcja SELECT – zagnieżdżenie

- Podaj nazwiska klientów, którzy złożyli zamówienie po 1 marca 2021:

```
SELECT DISTINCT nazwisko
FROM klient K, zamowienie
WHERE K.nr = klient_nr AND data_zlozenia > '2021-3-1'
```

  - 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 (GROUP BY)

Instrukcja SELECT – zagnieżdżenie, c.d.

- Właściwe rozwiązanie:

```
SELECT nazwisko FROM klient
WHERE nr IN ( SELECT klient_nr
              FROM zamowienie
              WHERE data_zlozenia > '2021-3-1'
            )
```

  - 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
           )
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 złożyli zamówienie po 1 marca 2021 – jeszcze jedno rozwiązanie:

```
SELECT nazwisko
FROM klient K
WHERE EXISTS (
   SELECT *
   FROM zamowienie
   WHERE K.nr = klient_nr AND data_zlozenia > '2021-3-1'
)
```

  - 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 złożyli zamówienie po 1 marca 2021:

```
SELECT DISTINCT nazwisko
FROM klient K, zamowienie
WHERE K.nr = klient_nr AND data_zlozenia > '2021-3-1'
```

- Podaj nazwiska klientów, którzy nie złożyli zamówienia po 1 marca 2021 ?
  - nie wiadomo, któremu warunkowi zaprzeczyć

```
data_zlozenia <= '2021-3-1'
```

  - 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 złożyli zamówienia po 1 marca 2021:

```
SELECT nazwisko FROM klient
WHERE nr NOT IN ( SELECT klient_nr FROM zamowienie
                  WHERE data_zlozenia > '2021-3-1' )
```

  - albo

```
SELECT nazwisko FROM klient K
WHERE NOT EXISTS (
  SELECT * FROM zamowienie
  WHERE K.nr = klient_nr AND data_zlozenia > '2021-3-1' )
```

- Będą jeszcze inne rozwiązania tego problemu

Instrukcja SELECT – operacje algebry relacji

- Podaj nazwiska klientów, którzy nie złożyli zamówienia po 1 marca 2021 ?

```
SELECT nazwisko FROM klient
EXCEPT
SELECT nazwisko FROM klient
WHERE nr IN ( SELECT klient_nr FROM zamowienie
              WHERE data_zlozenia > '2021-3-1' )
```

  - 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 wszelkie informacje o zamówieniach składanych przez klientów z Gdańska

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

  - 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
- Usuń dane o klientach z Gdańska

```
DELETE FROM klient
WHERE miasto = 'Gdańsk'
```