Bazy danych 2022: rozwiązania zadań z listy nr 3

Bartosz Brzostowski

19 kwietnia 2022

Często istnieje istnieje kilka istotnie różnych, równie dobrych rozwiązań tego samego zadania. Poniższe rozwiązania są przykładowe. Jeżeli jednak Twoje rozwiązanie bardzo różni się od przykładowego, a zwłaszcza jeśli daje różne wyniki, zastanów się, na czym polega różnica i ew. skonsultuj się z prowadzącym.

Podane rozwiązania nie wykorzystują, zgodnie z dodatkowym wymaganiem dla tej listy, podzapytań – dla wielu zadań istnieją alternatywne rozwiązania wykorzystujące podzapytania właśnie.

Ważna uwaga ogólna: kiedy mamy coś zgrupować np. względem produktów lub klientów, to najwłaściwszym kryterium grupowania jest tu ich identyfikator (`idp` lub `idk`). Intuicyjnie bardziej nasuwa się np. nazwa, ale jeśli może się ona powtarzać (a w naszej bazie może, bo nie ma na tej kolumnie więzu `UNIQUE`), to grupowanie po nazwie potencjalnie może „skleić" dwa różne produkty lub dwóch różnych klientów o tej samej nazwie – jest to może nienaturalne, ale zasadniczo możliwe.

Inne dodatkowe wymaganie na tej liście mówiło o tym, by wszystkie wyniki zapytań (poza tymi jednowierszowymi) „rozsądnie" sortować – co zostało w wielu przypadkach zignorowane. Jedną z przyczyn może być fakt, że domyślnym zachowaniem MySQL jest, by przy grupowaniu również sortować wyniki po tych samych kolumnach, po których jest grupowanie – wtedy np. przy grupowaniu po nazwach klientów / produktów od razu mamy „rozsądne" sortowanie. Po pierwsze wiemy już jednak, że (zazwyczaj) lepiej nie grupować po nazwie – a sortowanie po ID dla końcowego użytkownika raczej nie jest „rozsądne", po drugie natomiast to zachowanie MySQL nie jest gwarantowane standardem i dobrze jest zadeklarować sortowanie (choćby takie samo) jawnie za pomocą klauzuli `ORDER BY`.

Przypomnienie: tam, gdzie w klauzuli `GROUP BY` czy `ORDER BY` występuje stała liczbowa (np. `GROUP BY 1`), oznacza to odpowiednią kolumnę z klauzuli `SELECT` (która może być potencjalnie zadana skomplikowanym wyrażeniem, którego może nie chcemy po raz kolejny pisać). Jest to zachowanie podobne do aliasów, dostępne dlatego, że grupowanie czy porządkowanie po wartościach stałych nie ma specjalnego sensu.

Zadanie 1

```
SELECT SUM(cena * ilosc) FROM produkty;
```

Zadanie 2

```
SELECT GROUP_CONCAT(nazwa ORDER BY 1 SEPARATOR ", ") FROM klienci;
```

Tu oczywiście sortowanie musi być zadeklarowane w argumentach funkcji agregującej – w zapytaniu nie ma ono sensu, bo i tak chcemy otrzymać jeden wiersz.

Zadanie 3

```
SELECT DATEDIFF(MAX(data), MIN(data)),
    COUNT(DISTINCT idp),
    COUNT(DISTINCT idz)
  FROM zamow
    JOIN detal_zamow ON idz = z_id
    JOIN produkty ON idp = p_id
  WHERE nazwa LIKE "%Samsung%";
```

Ważne jest, żeby użyć bibliotecznej funkcji `DATEDIFF` – inaczej niż w PostgreSQLu, odejmowanie dat działa dosyć dziwacznie: daty są najpierw konwertowane na liczby tak, jakby były stringami. Ponieważ w tym przykładzie odejmowane daty różnią się o dwa miesiące i sześć dni, wynikiem odejmowania jest 206, podczas gdy poprawna liczba dni to 67.

Zadanie 4

```
SELECT DAYNAME(data), COUNT(DISTINCT k_id)
  FROM zamow
  GROUP BY 1
  ORDER BY 2 DESC;
```

Jest to jeden z przykładów na to, że można grupować nie tylko po wartościach kolumn, ale też wartościach funkcji wywołanych na wartościach kolumn. Tu, jak i w wielu innych miejscach (choć nie zawsze!), używamy `COUNT(DISTINCT ...)` – bez tego każdy obiekt (tu: klient) liczyłby się tyle razy, ile występuje w wyrażeniu kolumnowym, a mamy policzyć każdy tylko raz.

Zadanie 5

```
SELECT YEAR(data), MONTH(data), SUM(cena * sztuk)
  FROM zamow
    JOIN detal_zamow ON idz = z_id
    JOIN produkty ON idp = p_id
  GROUP BY 1, 2
  ORDER BY 1, 2;
```

Przykład grupowania względem więcej niż jednej kolumny na raz.

Zadanie 6

```
SELECT CEILING(cena/1000)*1000, JSON_ARRAYAGG(nazwa)
```

```
  FROM produkty
  GROUP BY 1
  ORDER BY 1;
```

Zadanie 7

```
SELECT nazwa
  FROM produkty
    JOIN detal_zamow ON idp = p_id
  GROUP BY idp
  HAVING SUM(cena * sztuk) > 7000
  ORDER BY 1 ASC;
```

Zadanie 8

```
SELECT klienci.nazwa
  FROM klienci
    JOIN zamow ON idk = k_id
    JOIN detal_zamow ON idz = z_id
    JOIN produkty ON idp = p_id
  WHERE cena > 1800
  GROUP BY idk
  HAVING SUM(sztuk) > 1
  ORDER BY 1 ASC;
```

Przykład zapytania, w którym występuje zarówno klauzula `WHERE`, jak i `HAVING`. Chcemy zagregować (policzyć) tylko niektóre produkty (i dlatego filtrowanie we `WHERE`), a po zagregowaniu – wyświetlić tylko niektóre grupy (stąd `HAVING`).

Zadanie 9

```
SELECT nazwa, AVG(sztuk)
  FROM produkty
    JOIN detal_zamow ON idp = p_id
    JOIN zamow ON idz = z_id
  GROUP BY idp
  HAVING BIT_OR(WEEKDAY(data) = 4)
  ORDER BY 2 DESC;
```

Popatrz na podpowiedź do tego zadania. W każdej grupie (związanej z danym produktem) sprawdzamy, czy daty są piątkowe, czy nie (`WEEKDAY(data) = 4`), a potem wartości tych testów agregujemy spójnikiem „lub” (`BIT_OR()`), żeby zachować tylko grupy, w których choć jedna data była piątkowa.

Zadanie 10

```
SELECT nazwa, COUNT(DISTINCT idz)
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
  GROUP BY idk
  ORDER BY 2 DESC;
```

Chcemy zachować wszystkich klientów, stąd złączenie zewnętrzne. `COUNT(DISTINCT ...)` dla grupy z samym NULLem w zliczanej kolumnie zwróci 0, zgodnie z naszym oczekiwaniem.

Zadanie 11

```
SELECT nazwa, IFNULL(SUM(sztuk), 0)
  FROM produkty
    LEFT JOIN detal_zamow ON idp = p_id
  GROUP BY idp
  ORDER BY 2 DESC;
```

Z kolei `SUM()` dla niezamawianego produktu zwróci NULL, więc musimy go jakoś obsłużyć.

Zadanie 12

```
SELECT miasto, IFNULL(SUM(sztuk), 0)
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
    LEFT JOIN detal_zamow ON idz = z_id
  GROUP BY 1
  ORDER BY 2 DESC;
```

Wprawdzie w bieżącym stanie bazy nie ma miejscowości, z których nic jeszcze nie było zamawiane – ale na wypadek, gdyby były, potrzebne jest złączenie zewnętrzne.

Zadanie 13

```
SELECT klienci.nazwa, miasto, IFNULL(SUM(cena * sztuk), 0)
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
    LEFT JOIN detal_zamow ON idz = z_id
    LEFT JOIN produkty ON idp = p_id
  GROUP BY idk
  ORDER BY 3 DESC;
```

Zadanie 14

```
SELECT data, SUM(cena * sztuk)
  FROM zamow
    LEFT JOIN detal_zamow ON idz = z_id
    LEFT JOIN produkty ON idp = p_id
  GROUP BY idz
  ORDER BY 2 DESC;
```

Tutaj, inaczej niż w większości innych zadań, grupujemy po zamówieniach – ale wciąż robimy to po ich identyfikatorach, a nie np. po datach, mimo że to daty mamy wyświetlić.

Zadanie 15

To jest trudne zadanie (mimo wskazówki), więc prześledźmy, jak dojść do jego rozwiązania. Zapomnijmy na chwilę o ograniczeniu na ceny produktów – wyświetlmy nazwy wszystkich klientów i liczby różnych zamówionych przez każdego z nich produktów. To nie powinno być skomplikowane:

```
SELECT klienci.nazwa, COUNT(DISTINCT idp)
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
    LEFT JOIN detal_zamow ON idz = z_id
    LEFT JOIN produkty ON idp = p_id
  GROUP BY idk
  ORDER BY 2 DESC;
```

(moglibyśmy nawet zrezygnować z ostatniego złączenia i zliczać `p_id`). Chcemy jednak liczyć tylko produkty droższe niż 1500 zł. Co się stanie, gdy dodamy klauzulę `WHERE cena > 1500`? Nic dobrego: odpadnie klient, który nic nie zamawiał (i cały trud pisania złączeń zewnętrznych na marne!).

No to może `WHERE cena > 1500 OR cena IS NULL`? To da dobry wynik, ale tylko w tym stanie bazy: istnieją klienci, którzy zamawiali tylko produkty tańsze niż 1600 zł, więc gdyby przyjąć taką kwotę jako ograniczenie, to zapytanie z taką klauzulą `WHERE` wprawdzie zachowałoby klienta, który nie zamawiał nic, ale wyrzuci klientów, którzy zamawiali tylko tanie produkty.

Wróćmy więc do wskazówki, która mówi nam, by zmodyfikować zapytanie do takiego:

```
SELECT klienci.nazwa, COUNT(DISTINCT idp)
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
    LEFT JOIN detal_zamow ON idz = z_id
    LEFT JOIN produkty ON (idp = p_id AND cena > 1500)
  GROUP BY idk
  ORDER BY 2 DESC;
```

i zastanówmy się, co dokładnie policzy się w ostatnim złączeniu. Dla klientów, którzy zamawiali drogie rzeczy, nie zmieni się nic w stosunku do zapytania z klauzulą `WHERE`:

połączą się tymi z wierszami tabeli `produkty`, których `cena` jest dostatecznie duża. Tak wyglądałby cały wynik, gdyby to złączenie było wewnętrzne.

Co innego klienci, którzy zamawiali tylko rzeczy tanie: dla nich żadne pary wierszy nie „przeżyją" warunku `idp = p_id AND cena > 1500` (tj. nawet, jeśli bywa spełniony pierwszy, to drugi już nie), więc – zgodnie z zasadą działania złączenia zewnętrznego – wiersze im odpowiadające, odpowiednio przedłużone NULLami, zostaną dołączone do wyżej opisanego wyniku złączenia wewnętrznego. Tak że nadal wszyscy klienci pozostaną w wyniku, ale ci, którzy nie zamawiali nic drogiego, będą mieć w kolumnie `idp` NULL. I to jest dokładnie to, o co nam chodzi.

Zadanie 16

```
SELECT produkty.nazwa, COUNT(DISTINCT idk)
  FROM produkty
    LEFT JOIN detal_zamow ON idp = p_id
    LEFT JOIN zamow ON idz = z_id
    LEFT JOIN klienci ON (idk = k_id AND miasto LIKE "W%")
  GROUP BY idp
  ORDER BY 2 DESC;
```

Zadanie analogiczne do poprzedniego.

Zadanie 17

```
SELECT z1.*
  FROM zamow z1
    JOIN zamow z2 ON z1.data <= z2.data
  GROUP BY z1.idz
  HAVING COUNT(z2.idz) <= 3
  ORDER BY data;
```

Zasada jest podobna do wzorcowego rozwiązania zadania 17 z listy 2 opisanego wcześniej. Jedna różnica jest taka, że tutaj wybieramy trzy skrajne wiersze, zamiast jednego. Druga to to, że poprzednio używaliśmy złączenia zewnętrznego – ale moglibyśmy tak zrobić i tutaj, tylko nierówność w `HAVING` należałoby zamienić na ostrą.