Bazy danych 2022 rozwiązania zadań z listy nr 4

Bartosz Brzostowski

4 maja 2022

Często istnieje istnieje kilka istotnie różnych, równie dobrych rozwiązań tego samego zadania – jest to szczególnie prawdziwe dla zadań z tej listy, gdzie wykorzystujemy już właściwie pełne możliwości DQL. 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.

Wszystkie rozwiązania w tym zestawie korzystają wyłącznie z perspektyw zdefiniowanych w zad. 1 bądź w danym zadaniu. Perspektywy były definiowane tylko dla tych podzapytań, do których odwoływano się więcej niż raz. Oczywiście kwestią stylu i indywidualnego wyboru jest używanie perspektyw mniej lub bardziej licznie.

Zadanie 1

```
CREATE VIEW wszyscy_klienci AS
SELECT idk, klienci.nazwa as knazwa,
    miasto, adres, telefon, zamow.*,
    detal_zamow.*, idp,
    produkty.nazwa as pnazwa,
    cena, ilosc
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
    LEFT JOIN detal_zamow ON idz = z_id
    LEFT JOIN produkty ON idp = p_id;
```

```
CREATE VIEW wszystkie_produkty AS
SELECT idk, klienci.nazwa as knazwa,
    miasto, adres, telefon, zamow.*,
    detal_zamow.*, idp,
    produkty.nazwa as pnazwa,
    cena, ilosc
  FROM klienci
    RIGHT JOIN zamow ON idk = k_id
    RIGHT JOIN detal_zamow ON idz = z_id
    RIGHT JOIN produkty ON idp = p_id;
```

Obie perspektywy łączą wszystkie cztery tabele bazy i różnią się tylko „kierunkiem” złączeń zewnętrznych. Przy okazji przekonujemy się o pewnym (zrozumiałym) ograniczeniu, jakiemu podlegają perspektywy: tak jak w tabelach, nie mogą się w nich powtarzać nazwy kolumn, więc jeśli takie coś zachodzi w definiującym perspektywę zapytaniu, trzeba jeszcze te kolidujące kolumny zaaliasować.

Zadanie 2

Porównaj te rozwiązania z pojawiającymi się w materiałach do poprzednich list. Logika rozwiązań pozostaje niezmieniona, ale skraca się ich forma (bo „zamiatamy ją pod dywan”, tj. do perspektywy).

Lista 2, zadanie 9

W tym i następnym zadaniu nie ma znaczenia, z której z perspektyw z zad. 1 skorzystamy – niezamówione produkty czy niezamawiający klienci na pewno nie wchodzą do wyniku.

```
SELECT adres
  FROM wszystkie_produkty
  WHERE pnazwa LIKE "%laptop%"
  GROUP BY idk
  ORDER BY REVERSE(knazwa);
```

Lista 2, zadanie 10

```
SELECT pnazwa
  FROM wszystkie_produkty
  WHERE telefon LIKE "%4%"
  GROUP BY idp
  ORDER BY cena;
```

Lista 2, zadanie 13

W tym i następnym zadaniu jest wprost napisane, że mamy wypisać wszystkie produkty, więc lepiej użyć perspektywy `wszystkie_produkty`.

```
SELECT pnazwa, miasto FROM wszystkie_produkty GROUP BY idp, miasto;
```

Lista 2, zadanie 14

```
SELECT pnazwa FROM wszystkie_produkty WHERE p_id IS NULL;
```

Lista 2, zadanie 15

A tu jest na odwrót.

```
SELECT knazwa FROM wszyscy_klienci WHERE z_id IS NULL;
```

Lista 3, zadanie 16

Gdybyśmy chcieli użyć tu którejkolwiek z perspektyw, nadawałaby się bardziej `wszystkie_produkty`, ale napotkalibyśmy na dokładnie te same problemy, co i bez niej, więc trudno powiedzieć, żeby szczególnie nam ona coś upraszczała. Skoro już możemy używać podzapytań, lepiej skorzystać z tej możliwości – patrz rozwiązanie do zad. 20 z tej listy.

Zadanie 3

```
SELECT z_lapt.knazwa
  FROM wszyscy_klienci AS z_lapt
```

```
    JOIN wszyscy_klienci AS z_tabl ON (z_lapt.idk = z_tabl.idk)
  WHERE z_lapt.pnazwa = "laptop"
    AND z_tabl.pnazwa = "tablet"
  GROUP BY z_lapt.idk;
```

To jest przykład poważnej oszczędności w objętości zapytania (i możliwości popełnienia błędów w nim) dzięki perspektywom. Bez nich musielibyśmy dwukrotnie powtórzyć jako podzapytanie fragment kodu definiujący złączenie czterech tabel.

Zadanie 4

```
SELECT knazwa
  FROM wszyscy_klienci
  WHERE cena > ANY (
    SELECT cena
      FROM produkty
      WHERE nazwa LIKE "%laptop%"
    )
  GROUP BY idk;
```

```
SELECT knazwa
  FROM wszyscy_klienci
  WHERE cena > (
    SELECT MIN(cena)
      FROM produkty
      WHERE nazwa LIKE "%laptop%"
    )
  GROUP BY idk;
```

Bardzo podobne rozwiązania – jedno używa podzapytania tablicowego z operatorem `ANY`, drugie – wierszowego z funkcją agregującą `MIN`. Nie wszystkie zapytania typu tego po lewej da się łatwo przekształcić do takiego, jak po prawej, ale te prostsze się da.

Zadanie 5

```
SELECT pnazwa
  FROM wszystkie_produkty
  WHERE knazwa < ALL (
    SELECT nazwa
      FROM klienci
      WHERE miasto = "Warszawa"
    )
  GROUP BY idp;
```

```
SELECT pnazwa
  FROM wszystkie_produkty
  WHERE knazwa < (
    SELECT MIN(nazwa)
      FROM klienci
      WHERE miasto = "Warszawa"
    )
  GROUP BY idp;
```

Podobnie jak w poprzednim zadaniu.

Zadanie 6

```
SELECT knazwa
  FROM wszyscy_klienci
  GROUP BY idk
  HAVING SUM(cena * sztuk) > (
    SELECT IFNULL(SUM(cena * sztuk), 0)
      FROM wszyscy_klienci
      WHERE knazwa = "Astro"
  );
```

Podzapytanie wylicza łączną wartość zamówień wyróżnionego klienta, nadzapytanie porównuje z nią łączne wartości zamówień wszystkich klientów. Użycie `IFNULL` jest tu trochę przesadne, na wypadek, gdyby klient „Astro” nie miał żadnych zamówień. Korzystamy z faktu, że jest tylko jeden klient o tej nazwie – gdyby było inaczej, zadanie byłoby źle zdefiniowane, a to zapytanie skutkowałoby błędem wykonania.

Zadanie 7

Rozwiązania podpunktów a) i b) – bardzo podobne, w b) musimy tylko wykombinować, jak liczyć średnią ważoną, kiedy nie mamy takiej funkcji bibliotecznej. Podzapytanie wierszowe wyznacza zadaną wartość, od której dwukrotności „odcinamy” wyświetlane w nadzapytaniu produkty.

```
SELECT nazwa
  FROM produkty
  WHERE cena > 2 * (
      SELECT AVG(cena)
        FROM produkty
    );
```

```
SELECT nazwa
  FROM produkty
  WHERE cena > 2 * (
      SELECT SUM(cena * ilosc)/SUM(ilosc)
        FROM produkty
    );
```

W podpunkcie c) robi się trochę gorzej, tj. musimy wykonać agregację w tabeli `detal_zamow`, żeby uzyskać wagi do średniej, stąd trzeci poziom zagnieżdżenia zapytań. Niezmienna pozostaje struktura zewnętrznego zapytania.

```
SELECT nazwa
  FROM produkty
  WHERE cena > 2 * (
      SELECT SUM(cena * lacznie)/SUM(lacznie)
        FROM (
            SELECT cena, SUM(sztuk) AS lacznie
              FROM produkty
                LEFT JOIN detal_zamow ON idp = p_id
              GROUP BY idp
          ) AS foo
    );
```

Zadanie 8

```
SELECT YEAR(data), WEEKOFYEAR(data), AVG(wartosc)
  FROM (
    SELECT data, SUM(cena * sztuk) AS wartosc
      FROM zamow
        JOIN detal_zamow ON idz = z_id
        JOIN produkty ON idp = p_id
      GROUP BY idz
    ) AS foo
  GROUP BY 1, 2;
```

Robimy dokładnie to, co we wskazówce: w podzapytaniu wyznaczamy wartość wszystkich

zamówień (`GROUP BY idz`), w nadzapytaniu grupujemy je po poszczególnych tygodniach (moglibyśmy też wykorzystać funkcję `YEARWEEK` jako pojedynczą kolumnę zamiast pierwszych dwóch w `SELECT` i `GROUP BY`) i liczymy średnią.

Wariant dla purystów, uwzględniający to, że puste zamówienia, gdyby istniały, powinny wejść do średniej z wartością 0: w podzapytaniu zastąpić oba złączenia przez `LEFT JOIN` i `SELECT`ować `IFNULL(SUM(cena * sztuk), 0)`.

Zadanie 9

```
CREATE VIEW wartosci_zamowien_klientow AS
SELECT idk, knazwa AS nazwa, IFNULL(SUM(cena * sztuk), 0) AS wartosc
  FROM wszyscy_klienci
  GROUP BY idk;

SELECT nazwa FROM
  wartosci_zamowien_klientow
  WHERE wartosc >= ALL (
      SELECT wartosc
        FROM wartosci_zamowien_klientow
    );
```

Tu znowu definiujemy nową perspektywę, żeby nie powtarzać takich samych podzapytań (tj. odwołujemy się do niej dwa razy). W tym rozwiązaniu nałożenie `IFNULL` na `SUM` jest dosyć istotne – w perspektywie `wszyscy_klienci`, o którą się opieramy, są też klienci, którzy nic nie zamawiają i bez tego mieliby w kolumnie `wartosc` naszej nowej perspektywy NULL. W takiej sytuacji skomplikowany robi się sens porównania `>= ALL`, które dotyczy zbioru zawierającego NULL i wymaga rozważenia w kategoriach logiki trójwartościowej. Dlatego dobrze (a także elegancko ze względu na „sens" definiowanej perspektywy) pozbyć się tych problematycznych NULLi.

Można też dokonać zamiany warunku (w stylu zadań 4–5) na `... WHERE wartosc >= (SELECT MAX(wartosc) FROM ...` co też rozwiąże problem, tj. wzięcie `MAX` ze zbioru zawierającego NULL daje bardziej przewidywalny rezultat.

Zadanie 10

Podpodzapytanie zlicza (w zależności od podpunktu) liczbę zamówień lub łącznie zamówionych sztuk każdego produktu, podzapytanie wybiera z tego maksimum, nadzapytanie używa tej wartości do przefiltrowania wyniku – tyle że w klauzuli `HAVING`, bo filtrowanie dotyczy wyniku agregacji. Różnice między podpunktami ograniczają się do dwóch linijek.

```
SELECT nazwa
  FROM produkty
    JOIN detal_zamow ON idp = p_id
  GROUP BY idp
  HAVING COUNT(DISTINCT z_id) = (
    SELECT MAX(iler)
      FROM (
        SELECT COUNT(DISTINCT z_id) AS iler
          FROM detal_zamow
          GROUP BY p_id
        ) AS foo
    );
```

```
SELECT nazwa
  FROM produkty
    JOIN detal_zamow ON idp = p_id
  GROUP BY idp
  HAVING SUM(sztuk) = (
    SELECT MAX(iles)
      FROM (
        SELECT SUM(sztuk) AS iles
          FROM detal_zamow
          GROUP BY p_id
        ) AS foo
    );
```

Tutaj użycie podzapytań tablicowych pozwala skrócić zapytanie i pozbyć się jednego poziomu zagnieżdżenia:

```
SELECT nazwa
  FROM produkty
    JOIN detal_zamow ON idp = p_id
  GROUP BY idp
  HAVING COUNT(DISTINCT z_id) >= ALL (
      SELECT COUNT(DISTINCT z_id)
        FROM detal_zamow
        GROUP BY p_id
    );
```

```
SELECT nazwa
  FROM produkty
    JOIN detal_zamow ON idp = p_id
  GROUP BY idp
  HAVING SUM(sztuk) >= ALL (
      SELECT SUM(sztuk)
        FROM detal_zamow
        GROUP BY p_id
    );
```

Zadanie 11

```
CREATE VIEW produkty_w_dniach AS
SELECT idp, nazwa, DATE(data) AS dzien, SUM(cena * sztuk) AS wartosc
  FROM produkty
    JOIN detal_zamow ON idp = p_id
    JOIN zamow ON idz = z_id
  GROUP BY 1, 3;

SELECT nazwa, dzien
  FROM produkty_w_dniach AS super
  WHERE wartosc = (
      SELECT MAX(wartosc)
        FROM produkty_w_dniach
        WHERE dzien = super.dzien
    );
```

Perspektywa zestawia łączne wartości zamówień na każdy produkt w każdym dniu. W podzapytaniu wybieramy największą z tych wartości danego dnia, w nadzapytaniu wybieramy ten produkt, który realizuje to maksimum. Podzapytanie jest skorelowane, odwołuje się do wartości `super.dzien` z nadzapytania. Zamiast niego można zrobić złączenie zewnętrzne i agregację:

```
SELECT pwd1.nazwa, pwd1.dzien
  FROM produkty_w_dniach AS pwd1
```

```
    LEFT JOIN produkty_w_dniach AS pwd2
      ON (pwd1.dzien = pwd2.dzien AND pwd1.wartosc < pwd2.wartosc)
  WHERE pwd2.idp IS NULL;
```

Inne ciekawe rozwiązanie, z podzapytaniem, ale nieskorelowanym, jest takie:

```
SELECT nazwa, dzien
  FROM produkty_w_dniach
  WHERE (dzien, wartosc) IN (
      SELECT dzien, MAX(wartosc)
        FROM produkty_w_dniach
        GROUP BY dzien
    );
```

Wykorzystujemy tu operator `IN` dla wiersza, który składa się z więcej niż jednej komórki – oczywiście liczba kolumn w podzapytaniu musi pasować.

Zadanie 12

Podobnie jak w poprzednim, mamy kolejno: pomocniczą perspektywę, rozwiązanie z podzapytaniem skorelowanym, złączeniem dwóch egzemplarzy perspektywy, i z podzapytaniem nieskorelowanym.

```
CREATE VIEW ostatnie_zamowienia_klientow_z_miast AS
SELECT miasto, idk, nazwa, MAX(data) AS data
  FROM klienci
    LEFT JOIN zamow ON idk = k_id
  GROUP BY 1, 2;

SELECT miasto, nazwa
  FROM ostatnie_zamowienia_klientow_z_miast AS super
  WHERE data = (
      SELECT MAX(data)
        FROM ostatnie_zamowienia_klientow_z_miast
        WHERE miasto = super.miasto
    );

SELECT ozkm1.miasto, ozkm1.nazwa
  FROM ostatnie_zamowienia_klientow_z_miast AS ozkm1
    JOIN ostatnie_zamowienia_klientow_z_miast AS ozkm2
      ON (ozkm1.miasto = ozkm2.miasto AND ozkm1.data <= ozkm2.data)
  GROUP BY ozkm1.miasto, ozkm1.idk
  HAVING COUNT(DISTINCT ozkm2.data) = 1;

SELECT miasto, nazwa
  FROM ostatnie_zamowienia_klientow_z_miast
  WHERE (miasto, data) in (
      SELECT miasto, MAX(data)
        FROM ostatnie_zamowienia_klientow_z_miast
```

```
        GROUP BY miasto
    );
```

Zadanie 13

I jeszcze raz analogicznie.

```
CREATE VIEW produkty_klientow AS
SELECT idk, knazwa, pnazwa, IFNULL(SUM(cena * sztuk), 0) AS wartosc
  FROM wszyscy_klienci
  GROUP BY 1, idp;

SELECT knazwa, pnazwa
  FROM produkty_klientow AS super
  WHERE wartosc = (
      SELECT MAX(wartosc)
        FROM produkty_klientow
        WHERE idk = super.idk
    );

SELECT pk1.knazwa, pk1.pnazwa
  FROM produkty_klientow AS pk1
    LEFT JOIN produkty_klientow AS pk2
      ON (pk1.idk = pk2.idk AND pk1.wartosc < pk2.wartosc)
  WHERE pk2.idk IS NULL;

SELECT knazwa, pnazwa
  FROM produkty_klientow AS super
  WHERE (idk, wartosc) IN (
      SELECT idk, MAX(wartosc)
        FROM produkty_klientow
        GROUP BY idk
    );
```

Zadanie 14

Dwa równie dobre rozwiązania – wydaje się, że łatwiejsze do napisania i zrozumienia, niż te ze złączeniami zewnętrznymi z listy nr 2.

```
SELECT nazwa
  FROM produkty
  WHERE idp NOT IN (
      SELECT p_id
        FROM detal_zamow
    );
```

```
SELECT nazwa
  FROM produkty
  WHERE NOT EXISTS (
      SELECT 1
        FROM detal_zamow
        WHERE idp = p_id
    );
```

W wersji z `EXISTS` podzapytanie jest skorelowane mimo, że nie używamy żadnych aliasów – ale kolumna `idp` nie występuje w podzapytaniu, tylko w nadzapytaniu. Ponieważ ope-

rator `EXISTS` ignoruje kolumny zapytania, sprawdza tylko, czy są zwrócone jakieś wiersze, możemy jawnie wy`SELECT`ować np. stałą wartość 1.

Zadanie 15

Jak w poprzednim.

```
SELECT nazwa
  FROM klienci
  WHERE idk NOT IN (
      SELECT k_id
        FROM zamow
        WHERE idz IN (
            SELECT z_id
              FROM detal_zamow
          )
    );
```

```
SELECT nazwa
  FROM klienci
  WHERE NOT EXISTS (
      SELECT 1
        FROM zamow
        WHERE EXISTS (
            SELECT 1
              FROM detal_zamow
              WHERE idz = z_id
          )
          AND idk = k_id
    );
```

Zadanie 16

I jeszcze raz.

```
SELECT *
  FROM zamow
  WHERE idz NOT IN (
    SELECT z_id
      FROM detal_zamow
  );
```

```
SELECT *
  FROM zamow
  WHERE NOT EXISTS (
    SELECT 1
      FROM detal_zamow
      WHERE idz = z_id
  );
```

Zadanie 17

```
SELECT *
  FROM zamow AS super
  WHERE (
    SELECT COUNT(*)
      FROM zamow
      WHERE data > super.data
    ) < 3;
```

W skorelowanym podzapytaniu liczymy, ile zamówień jest nowszych od tego, które rozważane jest w nadzapytaniu – i żądamy, żeby liczba ta była mniejsza niż 3. Bez podzapytania skorelowanego jest rozwiązanie jak na liście 3. Pojawiło się też takie nietypowe i sprytne, ale trochę trudniejsze do przeczytania i zrozumienia rozwiązanie, również unikające podzapytania skorelowanego:

```
SELECT *
```

```
  FROM zamow
  WHERE data >= (
      SELECT SUBSTRING_INDEX(SUBSTRING_INDEX(
                GROUP_CONCAT(data ORDER BY data DESC), ",", 3), ",", -1)
        FROM zamow
  );
```

Podzapytanie wybiera trzecią największą datę z tabeli `zamow` korzystając z sortowania w funkcji `GROUP_CONCAT` i faktu, że daty w MySQLu nie zawierają przecinków funkcjonujących jako separator w operacji konkatenacji.

Zadanie 18

Moglibyśmy np. dokończyć rozwiązanie wzorcowe podane do listy nr 2 zgodnie ze wskazówką:

```
SELECT COUNT(DISTINCT z_id)
  FROM produkty AS p1
    LEFT JOIN produkty AS p2 ON p1.ilosc < p2.ilosc
    LEFT JOIN detal_zamow ON p1.idp = p_id
  WHERE p2.idp IS NULL
  GROUP BY p1.idp;
```

Ale, skoro znamy już podzapytania, możemy zacząć od nowa, po prostu przekładając treść zadania na SQL. W podpodzapytaniu sprawdzamy, jaka jest największa liczność towaru na stanie sklepu, w podzapytaniu wybieramy ten produkt, który występuje w takiej liczności. W nadzapytaniu wybieramy te detale zamówień, które dotyczą tego właśnie produktu, i zliczamy dla nich różne ID zamówień.

```
SELECT COUNT(DISTINCT z_id)
  FROM detal_zamow
  WHERE p_id IN (
    SELECT idp
      FROM produkty
      WHERE ilosc = (
          SELECT MAX(ilosc)
            FROM produkty
        )
    )
  GROUP BY p_id;
```

Możemy jeszcze trochę spłaszczyć strukturę zapytania stosując złączenie:

```
SELECT COUNT(DISTINCT z_id)
  FROM detal_zamow
    JOIN produkty ON idp = p_id
  WHERE ilosc = (
      SELECT MAX(ilosc)
        FROM produkty
```

```
    )
  GROUP BY p_id;
```

Zadanie 19

Zamiast stosować dziwne i trudno zrozumiałe warunki w złączeniach wewnętrznych, jak na liście nr 3, piszemy podzapytanie zwracające tylko drogie produkty i używamy go w złączeniu zewnętrznym w klauzuli `FROM`.

```
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 (
      SELECT *
        FROM produkty
        WHERE cena > 1500
      ) AS foo ON idp = p_id
  GROUP BY idk
  ORDER BY 2 DESC;
```

Zadanie 20

Analogicznie.

```
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 (
      SELECT *
        FROM klienci
        WHERE miasto LIKE "W%"
      ) AS foo ON idk = k_id
  GROUP BY idp
  ORDER BY 2 DESC;
```