Sprawdzian z SQL nr 4 26.05.2022

Dla każdego z poniższych zadań napisz odpowiednie polecenia SQL. Oczekujemy rozwiązania w postaci pliku zawierającego **treści** poleceń SQL, a nie znalezionej odpowiedzi. Nie będą sprawdzane jakiekolwiek zapytania niepoprawne składniowo — sprawdź swoje rozwiązanie używając `\i plik.sql` ! Plik możesz wysyłać wielokrotnie, sprawdzana będzie wyłącznie najnowsza wersja.

Wczytaj do swojej bazy danych plik `tpch-keys.sql`. Rozwiązania wysyłaj przez formularz pod adresem `https://hera.stud.ii:2015`. **Rób to jak najczęściej! Wszystkie dane na Waszych komputerach są kasowane po restarcie.**

Format pierwszej linijki rozwiązania: `-- grupa-imie-nazwisko`, gdzie `grupa` to inicjały prowadzącego Twoją grupę (`pwi`/`jotop`/`plg`/`mabi`/`akr`/`rfe`), np. `pwi-Jan-Kowalski`. Wymagany format całego pliku z rozwiązaniem:

```
-- grupa-imie-nazwisko
-- Zadanie 1
<zapytanie>

-- Zadanie 2
<zapytanie>
...
```

Zadanie 1 (2 pkt.) Dla każdego kraju podaj stan konta najbogatszego i najbiedniejszego klienta z niego pochodzącego (na podstawie `c_acctbal`). Wyniki posortuj alfabetycznie po nazwie kraju.

Rozwiązanie

```
SELECT n_name, MAX(c_acctbal), MIN(c_acctbal)
   FROM customer JOIN nation ON c_nationkey = n_nationkey
   GROUP BY n_nationkey
   ORDER BY 1;
```

Zapytanie wzorcowe zwraca 25 krotek 3-kolumnowych.

Zadanie 2 (3 pkt.) Chcielibyśmy zablokować możliwość usuwania zamówień z tabeli `orders` gdy odnoszą się do nich jakieś wpisy w tabeli `lineitem`. Wyjątkowo jednak dopuszczamy możliwość usuwania zamówień ze statusem `'O'` – przy próbie usunięcia takiego zamówienia wszystkie odnoszące się do niego wpisy powinny zostać usunięte kaskadowo.

Zrealizuj powyższy cel w następujący sposób:

- Zadeklaruj brakujący więz klucza obcego pomiędzy tabelami `orders` i `lineitem` w taki sposób, aby aktualizacja klucza w tabeli `orders` przenosiła się na tabelę `lineitem`, a usuwanie z tabeli `orders` nie było dozwolone, gdy odnoszą się do niej jakieś wpisy z tabeli `lineitem`.

- Napisz wyzwalacz dla tabeli `orders`, który pomimo obecności więzu z poprzedniego podpunktu umożliwi usuwanie wierszy tej tabeli, o ile spełniają one warunek `o_orderstatus = 'O'`. Działanie wyzwalacza powinno polegać na usuwaniu odpowiednich wierszy z tabeli `lineitem`.

Rozwiązanie

```
ALTER TABLE lineitem
   ADD FOREIGN KEY (l_orderkey) REFERENCES orders(o_orderkey)
   ON UPDATE CASCADE ON DELETE RESTRICT;

CREATE FUNCTION deleteOrderedLineitems()
   RETURNS TRIGGER AS $$
   BEGIN
      DELETE FROM lineitem
         WHERE l_orderkey = OLD.o_orderkey;
      RETURN OLD;
   END;
   $$ LANGUAGE PLPGSQL;

CREATE TRIGGER deleteOrderedCascade
   BEFORE DELETE ON orders
   FOR EACH ROW
   WHEN (OLD.o_orderstatus = 'O')
   -- oczywiście zamiast tego może być np. IF w funkcji
   EXECUTE FUNCTION deleteOrderedLineitems();
```

Tu można mieć ochotę stworzyć wyzwalacz `FOR EACH STATEMENT`, który jednym wyrażeniem `DELETE` usuwałby wpisy z `lineitem` dla wszystkich usuwanych wartości `o_orderkey` spełniających `o_orderstatus = 'O'`. Niestety, dla tego typu wyzwalaczy nie mamy jak się odwołać do wierszy, które próbuje się usunąć – zmienna `OLD` jest wtedy `NULL`em, zaś klauzula `REFERENCING` polecenia `CREATE TRIGGER` dozwolona jest tylko dla wyzwalaczy `AFTER`. W związku z tym, jeżeli liczymy na jakieś chytre optymalizacje kaskadowych usunięć po stronie DBMS, to mechanizm opisany w pierwszym akapicie treści zadania należałoby zrealizować komplementarnie – wybierając akcję integralności referencyjnej `ON DELETE CASCADE` i wyzwalaczem blokować usuwanie wierszy spełniających `o_orderstatus <> 'O'`, do których nic się nie odnosi w `lineitem`. Ale sprawdzanie tego ostatniego warunku w trybie `FOR EACH ROW` może budzić takie same wątpliwości, jak wcześniej...

Zadanie 3 (2 pkt.) Wypisz wszystkie pozycje `lineitem`, których `l_quantity` jest najwyższa w ramach danego zamówienia. Rozważ wyłącznie zamówienia o `orderkey` nie większym niż 100. Zignoruj pozycje, dla których `l_quantity` to `NULL`. Dla każdego wyniku wypisz dwie kolumny `l_orderkey` oraz `l_linenumber`. Wyniki

posortuj rosnąco względem pierwszej kolumny oraz w drugiej kolejności względem drugiej.

Rozwiązanie

```
SELECT l_orderkey, l_linenumber FROM lineitem l1
WHERE l_orderkey <= 100 AND l_quantity = (
      SELECT MAX(l_quantity)
         FROM lineitem l2
         WHERE l1.l_orderkey = l2.l_orderkey)
ORDER BY 1, 2;
```

Uwaga: może się zdarzyć, ze dla jakiegoś zamówienia będzie więcej niż jedna taka pozycja. Wzorcowe zapytanie zawraca 30 krotek (a nie np. 28!).

Zadanie 4 (3 pkt.) Dla wszystkich możliwych par regionów r_1, r_2 wypisz ich nazwy oraz łączną kwotę, którą klienci z r_1 zapłacili za części od dostawców z r_2. Wyznaczając ją, poza bazową ceną `l_extendedprice` uwzględnij zniżkę `l_discount` oraz podatek `l_tax`, przy czym zauważ, że obie te wartości to liczby dodatnie istotnie mniejsze od 1. Rezultat uporządkuj kolejno względem kwoty, nazwy r_1 i nazwy r_2.

Rozwiązanie

```
SELECT cr.r_name, sr.r_name,
      COALESCE(SUM(l_extendedprice * (1-l_discount) * (1+l_tax)), 0)
   FROM region AS sr CROSS JOIN region AS cr
      LEFT JOIN nation AS sn ON sr.r_regionkey = sn.n_regionkey
      LEFT JOIN nation AS cn ON cn.n_regionkey = cr.r_regionkey
      LEFT JOIN (lineitem
         JOIN orders ON l_orderkey = o_orderkey
         JOIN customer ON o_custkey = c_custkey
         JOIN supplier ON s_suppkey = l_suppkey
         ) AS foo ON c_nationkey = cn.n_nationkey
            AND sn.n_nationkey = s_nationkey
   GROUP BY cr.r_regionkey, sr.r_regionkey
   ORDER BY 3, 1, 2;
```

Gdyby można było pominąć regiony nie kupujące bądź nie sprzedające nic, zadanie byłoby tylko żmudne – wystarczałoby zwykłe wewnętrzne złączenie łańcuszka tabel z grupowaniem po parach identyfikatorów regionów i agregacją. Żeby w wyniku były jednak obecne wszystkie pary regionów, potrzebny jest kartezjański „kwadrat" tabeli `region`, do którego zewnętrznie dołączymy całą resztę, np. w formie podzapytania.

Jeśli ktoś miałby jednak *niezdrową obsesję unikania podzapytań*, to może czekać go niemiła niespodzianka. Ponieważ konieczne jest stosowanie złączeń zewnętrznych, to w pośrednich stadiach obliczeń pojawiać się będą wiersze z `NULL`ami, które nie będą miały wpływu na ostatecznie wyliczone wartości, ale może być ich całkiem sporo. Poprawne rozwiązanie bez podzapytań ani nawiasowań wylicza się w naszej bazie dobrych parę minut.

Uważny czytelnik może zapytać, dlaczego w powyższym rozwiązaniu również kopie tabeli `nation` nie zostały umieszczone wewnątrz podwyrażenia ze złączeniami wewnętrznymi. Otóż problemem jest jednoznaczność nazw kolumn i zasięg widoczności aliasów: aliasy zadeklarowane w zanawiasowanym podwyrażeniu nie są już widoczne na zewnątrz nawiasów (*ERROR: invalid reference to FROM-clause entry for table "sn" [...] HINT: There is an entry for table "sn", but it cannot be referenced from this part of the query*). A więc jeśli, tak jak byłoby w tej sytuacji, w podwyrażeniu są dwie kopie tej samej tabeli (oczywiście z różnymi aliasami), to na zewnątrz i tak nie ma możliwości jednoznacznego odniesienia się do ich kolumn. A żeby wprowadzić aliasy samych kolumn, potrzebne jest podzapytanie...

Po dodaniu szóstego regionu, np. poleceniem

```
INSERT INTO region VALUES (420, 'MOON', NULL),
```

wzorcowe zapytanie zwraca 36 krotek nie zawierających `NULL`i.