Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Predmet / Subject

– Databázové systémy / Database systems –

- Dokumentácia / Documentation -

Zadanie č.3

Ak. Rok / Academic term: 2022/2023, letný semester

Cvičiaci / Instructors:
Ing. Jakub Dubec

Študent / Student:
Adam Grík

Bratislava, 2023.

Obsah

Úvod

Úlohou tohto zadania bolo implementovať HTTP end-pointy podľa popisu zo zadania. Aplikácia číta dáta zo zadaného datasetu flights.sql. Riešenie implementujem v programovacom jazyku Python. Obsahom tejto dokumentácie je popis jednotlivých SQL dopytov.

Endpoint 1

```
SELECT
        sub.seat_no,
        COUNT(sub.seat_no) as "count"
FROM (
        SELECT
                seat_no,
                DENSE_RANK() OVER (PARTITION BY flights.flight_id ORDER BY
                bookings.book_date) as "rank"
        FROM bookings.boarding_passes
        JOIN bookings.tickets ON (boarding_passes.ticket_no = tickets.ticket_no)
        JOIN bookings.flights ON (boarding_passes.flight_id = flights.flight_id)
        JOIN bookings.bookings ON (tickets.book_ref = bookings.book_ref)
        WHERE aircraft_code = %s
        ) AS sub
WHERE sub."rank" = %s
GROUP BY sub.seat_no
ORDER BY "count" DESC
LIMIT 1;
```

Najdôležitejšou časťou tohto SQL dopytu je subquery s názvom sub. V tejto subquery si vyťahujem najprv *seat_no*, následne pomocou window funkcie *DENSE_RANK*, priradím ku každému sedadlu rank na základe toho kedy bolo toto sedadlo zarezervované konkrétnom *flight_id*. Funkcia *DENSE_ RANK* zabezpečuje to, že priraďuje rank tak, že ak sú dve sedadlá zarezervované v rovnaký čas, tak im priradí rovnaký rank, a pri ďalšej hodnote nevznikne gap. Následne v hlavnom *SELECT-e* už len zo subquery vyberám *seat_no* a pomocou funkcie *COUNT* spočítavam tieto sedadlá. Nakoniec treba ešte pridať podmienku s hodnotou *sub.”rank”*, ktorú získávame pomocou funkcie *DENSE_RANK* v subquery, a ktorá nám zabezpečuje to, že získame iba sedadlá, ktoré boli zarezervované ako k-te. Následne už to len zgrupneme podľa *seat_no,* zoradíme a určíme *LIMIT 1* aby sme získali len sedadlo ktoré bolo vybrané najčastejšie ako k-te.

Endpoint 2

```
SELECT
        tickets.ticket_no,
        tickets.passenger_name,
        array_agg(array[f.departure_airport, f.arrival_airport, f.flight_time,
        f.cumulative_flight_time] ORDER BY f.actual_departure)
FROM tickets
JOIN (SELECT
         passenger_name,
         tickets.ticket_no,
         ticket_flights.flight_id,
         departure_airport,
         arrival_airport,
         actual_departure,
         to_char((flights.actual_arrival - flights.actual_departure)::time,
        'FMHH24:MI:SS') AS flight_time,
        to_char((sum(flights.actual_arrival - flights.actual_departure) OVER
        (PARTITION BY passenger_name ORDER BY actual_departure)),
        'FMHH24:MI:SS')  as cumulative_flight_time
        FROM tickets
         JOIN ticket_flights on (tickets.ticket_no = ticket_flights.ticket_no)
         JOIN flights on (ticket_flights.flight_id = flights.flight_id)
        WHERE book_ref = %s
        GROUP BY tickets.ticket_no, passenger_name, ticket_flights.flight_id,
        flights.departure_airport, flights.arrival_airport, flights.actual_departure,
        flights.actual_arrival
       ) AS f ON f.ticket_no = tickets.ticket_no
WHERE book_ref = %s
GROUP BY tickets.ticket_no
ORDER by ticket_no
```

V tejto query získavam najprv stĺpce *ticket_no* a *passenger_name*, potom si vytváram dovojrozmerné pole do ktorého vkladám hodnoty získané zo subquery. V subquery robím to, že si najprv vytiahnem všetky potrebné stĺpce. Zo subquery potrebujem iba stĺpce *departure_airport* a *arrival_airport*, a tak isto aj strávený čas v lietadle. Dĺžku jednotlivých letov počítam jednoducho a to tak, že iba odpočítam *actual_arrival* od *actual_departure* a vložím do správneho formátu. Celkový nalietaný čas počítam pomocou window funkcie, výpočet je rovnaký ako v predošlom výpočte avšak toto celé dávam ešte do tzv. kumulatívneho sum-u, pomocou *window funkcie*. Funguje to tak, že tento sum rozdelím podľa *passenger_name* aby mi to počítalo iba vrámci jedného pasažiera a následne len zoradím podľa *actual_departure*, čo mi zabezpečí to, aby mi hodnotu *sum* pripočítaval priebežne v každom *SELECT-e*. Potrebné hodnoty z tejto subquery už len vložím do dvojrozmerného poľa v hlavnom *SELECT-e*. Nakoniec už len pridám potrebné podmienky, potrebné zgrupovanie a zoradím poďla zadania.

Endpoint 3

```
SELECT
        sub.seat_no,
        COUNT(sub.flight_id) as "count",
        array_agg(sub.flight_id ORDER BY flight_id)
FROM (
        SELECT
                seat_no,
                boarding_passes.flight_id as flight_id,
                DENSE_RANK() OVER (ORDER BY seat_no, boarding_passes.flight_id)
                as dense_rank_result,
                boarding_passes.flight_id – DENSE_RANK() OVER(ORDER BY seat_no,
                boarding_passes.flight_id ) as sequence_identifier
        FROM boarding_passes
        JOIN flights ON (flights.flight_id = boarding_passes.flight_id)
        WHERE flight_no = %s
        ) AS sub
GROUP BY sub.sequence_identifier, sub.seat_no
ORDER BY "count" DESC, sub.seat_no
LIMIT %s;
```

V tejto query je opäť najdôležitejšia časť subquery *sub*. V subquery *sub* najprv získavam stĺpce *seat_no* a *flight_id*. Následne si pomocou window funkcie *DENSE_RANK* priradím ku každému sedadlu hodnotu od 1 až po n. A ako posledné si v tejto subquery vyťahujem tzv. *sequence_identifier*, ktorý zapezpečuje to, že odpočítavam hodnotu priradenú pomocou *DENSE_RANKU*, ktorá je vysvetlená v predošlej vete, od hodnoty *flight_id*. Vyzerá to takto:

	seat_no character varying (4)	flight_id integer	dense_rank_result bigint	sequence_identifier bigint
1	18A	13069	1	13068
2	18A	13070	2	13068
3	18A	13071	3	13068

V tomto prípade sa výpočty vykonávajú napr. 13069 – 1, 13070 – 1 atď.. Ako môžeme vidieť zabezpečí mi to, to, že všetky sedadlá ktoré nasledujú po sebe (boli zarezervované vkuse po sebe) budú mať hodnotu stĺpca *sequence_identifier* rovnakú. To znamená, že vďaka tomu viem povedať že tieto tri sedadlá boli po sebe rezervované v daných zobrazených lietadlách. Následne už len v hlavom *SELECT-e* získavam stĺpce *seat_no*, pomocou funkcie *COUNT* spočítavam počet letov, v ktorých dané lietadlo bolo rezervované po sebe, a následne vkladám do poľa všetky *flight_id*, v ktorých bolo sedadlo rezervované po sebe. Nato aby som zabezpečil správne výsledky je na konci tejto query veľmi doležitý *GROUP BY sub.sequence_identifier*, ktorý mi zabezpečí to aby mi dané výsledky a počty zgruplo podľa spomínaného *sequence_identifier*, čo znamená že budem mať v jednom riadku len sedadlo, ktoré bolo rezervované v po sebe idúcich *flight_id* v mojom poli.

Endpoint 4

```
SELECT
        sum_amount::Integer as "amount", to_char("date", 'YYYY-MM') as "month",
        num_day
FROM (
 SELECT
        DATE_TRUNC('month', actual_departure) as "date",
        DATE_TRUNC('day', actual_departure) as "day",
        sum(amount) as sum_amount,
        extract(day from actual_departure) as "num_day",
        RANK() OVER (PARTITION BY DATE_TRUNC('month', actual_departure)
        ORDER BY SUM(amount) DESC) as rank
 FROM ticket_flights
 JOIN flights ON (flights.flight_id = ticket_flights.flight_id)
 WHERE aircraft_code = %s AND flights.actual_departure IS NOT NULL
 GROUP BY "date", "day", "num_day"
        ) sub
WHERE rank = 1
ORDER BY sum_amount DESC;
```

V tomto endpointe opäť využívam subquery *sub*. Hlavná funkcia tejto subqery je to, že si vďaka nej vytiahnem každý jeden deň v danom mesiaci, a tak isto aj jeho príjem vrámci zadaného lietadla. Funguje to následovne: príjem každého dňa si počítam jednoducho pomocou funkcie sum, následne ku každému dňu priradím rank na základe celkového príjmu v danom dni pomocou window funkcie *RANK OVER* a to tak, že rank rozdeľujem v rámci každého mesiaca. Následne už len pomocou *GROUP BY* to zgrupnem podľa potrebných hodnôt. Čo znamená že daná subquery mi vracia všetky dni, celkový príjem daného dňa, dátum daného dňa, poradie dňa z hľadiska dátumu v danom mesiaci a tak isto aj rank na základe celkového príjmu v dni vrámci mesiaca do ktorého daný deň patrí.

Následne už si v hlavnom *SELECT-e* vyťahujem potrebné údaje z danej subquery. A tie potrebné údaje sú: suma ako celkový príjem v danom dni, mesiac v ktorom sa daný deň nachádza a tak isto aj poradie dňa z hľadiska dátumu v danom mesiaci. Nakoniec som ešte pridal podmienku, ktorá mi zabezečí to, že mi *SELECT* vytiahne iba dni s rankom 1, čo znamená že získame iba najlepšie dni z hľadiska príjmu v danom mesiaci.