VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

Databázové systémy 2018

Dokumentácia projektu

Zadání č. 37 – Rezervace letenek

Andrej Naňo (xnanoa00), **Peter Marko** (xmarko15)

Brno, 1. květen 2018

Obsah

Úvod

Práca na projekte prebehla bez významných problémov. Prácu sme si vždy rozdelili podľa intuície a následne navzájom skontrolovali funkčnosť.

Zadanie

Zvolili sme si minuloročnú tému č. 37, „Rezervace letenek“.

Jej kompletné zadanie znie takto:

> Vaším úkolem je navrhnout webovou aplikaci pro rezervaci letenek. Systém musí uživateli umožnit specifikovat požadavek pomocí místa odletu a příletu, data, času, třídy, letecké společnosti apod. Jedno letadlo může létat na více letech, stejně tak mezi dvěmi destinacemi může létat více letadel různých společností. Zákazník si rezervuje letenku, která může být na několik letů, i různých společností (např. letenka Praha -> San Francisco s lety Praha -> Londýn ČSA, Londýn -> San Francisco British Airways). Systém musí umožnit klientovi tisk letového itineráře, který obsahuje informace o dobách příletů a odletů na jednotlivých letištích. Každý typ letadla má různý počet sedadel a jejich rozvržení do tříd. Nemodelujte jednotlivá sedadla. Rezervací může klient zamluvit i více míst v jednom letu. Technická mezipřistání modelovat nemusíte. Systém musí evidovat, která společnost kdy, odkud a kam létá. Cena sedadla v dané třídě může být u každé společnosti jiná, cena letenky je dána součtem cen všech rezervovaných sedadel na všech letech. Pokud klient rezervovanou letenku včas nezaplatí, rezervace se z databáze smaže.

Vytvorenie základných objektov schématu databáze

Bolo potrebné vytvoriť tabuľky:

airlines
airplanes
airports
flights
customers
passengers
reservations
tickets
search_records

Snažili sme sa na mieste primárnych kľúčov použiť na identifikáciu objektov reálne a v praxi používané formáty kódov, ktoré využívajú letecké spoločnosti po celom svete (IATA formát).

Ukážka formátov identifikátorov :

Letiská:	JFK
Letecké spoločnosti:	BA
Lety:	BA026
Letenky:	160-4837291830

V tabuľke „flights“ bolo potrebné zadefinovať údaje o prílete a odlete vo formáte: `TIMESTAMP WITH TIME ZONE`, keďže jednotlivé lety často začínajú a končia v iných časových pásmach. Čas sa teda vždy udáva ako čas v mieste príletu/odletu.

Naplnenie tabuliek vzorovými dátami

Našou snahou bolo naplniť tabuľky čo najviac réalnymi údajmi.
Používali sme teda reálne vyhľadávače letov (napr. *flights.google.com*), informácie ohľadom sedadiel na jednotlivých lietadlách, flotilách leteckých spoločností alebo ich domovských letiskách (*seatguru.com*, *wikipedia.com*).

Ukážkove príkazy výberu dát

Implementovali sme 7 príkazov typu SELECT.

2x JOIN 2 tables:

Funkcia: Zistiť, ktoré lety prevádzkuje letecká spoločnosť British Airways.

```
SELECT *
FROM flights NATURAL JOIN airlines
WHERE full_name = 'British Airways';
```

Pomocou NATURAL JOIN spojí tabuľky flights a airlines a vyberie len riadky, ktoré obsahujú spoločnosť s požadovaným názvom.

Funkcia: Zistiť, aké rôzne lietadlá vlastní letecká spoločnosť American Airlines.

```
SELECT DISTINCT producer, model
FROM airlines NATURAL JOIN airplanes
WHERE full_name = 'American Airlines';
```

Pomocou DISTINCT vyberie len jedinečné výsledky a vypíše rôzne typy lietadiel.

1x JOIN 3 tables:

Funkcia: Zistiť, ktoré letecké spoločnosti lietajú na trase Londýn -> New York.

```
SELECT DISTINCT full_name
FROM  flights NATURAL JOIN airlines, airports A1, airports A2
WHERE A1.city = 'London'
AND A2.city = 'New York'
AND flights.origin = A1.airport_code
AND flights.destination = A2.airport_code;
```

Použité sú dva typy spojení, NATURAL JOIN a CROSS JOIN, na spojenie letov, leteckých spoločností a letísk. Ďalej je špecifikované mesto, v ktorom sa letisko má nachádzať a na konci spojovacia podmienka pre kód počiatočného letiska a cieľového letiska.

2x GROUP BY + agregačná funkcia:

Funkcia: Vypísať všetky letecké spoločnosti zoradené podľa počtu destinácií, do ktorých ponúkajú lety.

```
SELECT full_name, COUNT(DISTINCT A.airport_code)
FROM flights NATURAL JOIN airlines, airports A
WHERE flights.destination = A.airport_code
GROUP BY full_name
ORDER BY 2 DESC;
```

Pomocou GROUP BY špecifikujeme, že chceme agregovať podľa názvov spoločností. ORDER BY špecifikuje, že treba tabuľku zoradiť podľa druhého stĺpca.

Funkcia: Zistiť, ktorí pasažieri majú zakúpené viac ako 2 letenky.

```
SELECT p.first_name, p.last_name, COUNT(DISTINCT t.ticket_number)
FROM  tickets t, passengers p
WHERE t.passenger = p.id
GROUP BY p.first_name, p.last_name
HAVING COUNT(DISTINCT t.ticket_number) > 2;
```

Pomocou CROSS JOIN sa prepoja tabuľky s lístkami a pasažiermi. Použije sa agregačná funkcia a na konci sa definuje, že nás zaujimajú len riadky kde je počet leteniek väčší ako 2.

1x EXISTS:

Funkcia: Zistiť, ktorá letecká spoločnosť prevádzkuje lety súčasne do miest Helsinki a New York.

```
SELECT airlines.full_name
FROM airlines
WHERE EXISTS (
    SELECT flight_number
    FROM flights
    WHERE airline = airlines.airline
    AND flights.destination IN ( SELECT airport_code FROM airports
                                 WHERE airports.city = 'Helsinki'))
AND EXISTS (
    SELECT flight_number
    FROM flights
    WHERE airline = airlines.airline
    AND flights.destination IN ( SELECT airport_code FROM airports
                                 WHERE airports.city = 'New York'));
```

Príkaz sa skladá z dvoch vnorených príkazov SELECT.
Prvý zisťuje, či spoločnosť lieta do Helsínk a druhý, či lieta do New Yorku.
Pomocou EXISTS zistí, či tabuľka obsahuje aspoň jeden riadok.

1x IN s vnoreným SELECT:

Funkcia: Vypísať všetky lety z Viedne do New Yorku.

```
SELECT flights.flight_number , flights.departure_time,
flights.arrival_time
FROM flights
WHERE flights.origin IN (
 SELECT airport_code FROM airports WHERE airports.city = 'Vienna')
AND flights.destination IN (
 SELECT airport_code FROM airports WHERE airports.city = 'New York');
```

Pomocou klauzule IN sa špecifikuje počiatočné a cieľové letisko. V jednom meste môžu byť viaceré letiská. Týmto spôsobom je možné špecifikovať lety medzi mestami a nie medzi letiskami.

Triggery

Implementovali sme tri typy triggerov.

Úlohou prvého typu triggeru je nastaviť nové id položky v tabuľke. Trigger sa spustí vždy pri pridaní novej položky, pre implementáciu sa využívajú sekvencie a pomocou NEXTVAL sa získava nové ID položky.

Druhý typ trigeru udržiava počty voľných sedadiel v jednotlivých triedach na daných letoch. Keď sa pridá nový záznam o letenke s určitou triedou sedadla, tak sa zníži počet voľných sedadiel v danej triede pre daný let.

Posledný typ triggeru taktiež slúži na nastavovanie počtu voľných sedadiel, ale iba pri vytváraní nového letu. Podľa lietadla použitého na konkrétny let sa nastaví počet voľných miest v triede podľa toho, koľko je maximálny počet sedadiel v danej triede pre konkrétne lietadlo.

Procedúry

Vytvorili sme dve procedúry:

```
customer_ticket_avg_cost()
airline_plane_percentage()
```

Procedúra **`customer_ticket_avg_cost()`** spočíta priemernú sumu, ktorú zákazník zaplatil za jednu letenku. Zákazník môže vytvoriť viacero rezervácií a každá rezervácia môže obsahovať viacero leteniek. Rezervácia môže obsahovať letenky pre rôznych pasažierov. Všetky letenky v rezervácii sa platia ako celok. Táto procedúra prechádza letenkami v rámci rezervácií konkrétneho zákazníka, inkrementuje počet leteniek a zvyšuje celkovú cenu. Na konci z týchto údajov vypočíta priemernú cenu za letenku. Pokiaľ zákazník nemá rezerváciu, alebo rezervácia neobsahuje žiadne letenky, tak sa volá výnimka.

Procedúra **`airline_plane_percentage()`** spočíta percento všetkých lietadiel v databáze, ktoré patria konkrétnej spoločnosti. Pokiaľ je tabuľka lietadiel prázdna, vyvolá sa výnimka. Procedúra prechádza všetkými lietadlami v tabuľke a pokiaľ lietadlo patrí určenej spoločnosti, tak sa inkrementuje čítač num_planes, čítač num_all_planes sa inkrementuje pri každom prechode. Na konci sa z týchto dvoch hodnôt spočíta percento a vypíše sa.

EXPLAIN PLAN a použitie INDEX

Úlohou príkazu EXPLAIN PLAN je zobrazenie postupnosti realizácie operácií optimalizátorom Oracle pre vybraný SQL príkaz. Okrem toho taktiež poskytne informácie o výkonnostnej cene pre každú operáciu a čas vykonania.

Pre príklad je použitý príkaz :

```
SELECT reservation, first_name, last_name,
      COUNT(*) AS num_of_flights, SUM(cost) AS cost_all
FROM passengers, tickets
WHERE passengers.id = tickets.passenger
GROUP BY reservation, first_name, last_name;
```

Jeho úlohou je zobraziť užitočné informácie o každom pasažierovi v rámci určitej rezervácie (tj. ak sa pasažier nachádza vo viacerých rezerváciach, tak sa môže v zozname vyskytnúť viackrát). Takéto zobrazenie môže mať zmysel pre zákaznika, ktorý by chcel mať prehľad o tom, koľko by mal každý pasažier v jeho rezervacii zaplatiť dokopy za všetky jeho letenky. Zobrazí sa taktiež počet letov, ktoré daného pasažiera čakajú, jeho celé meno a číslo rezervácie.

Ukážka výstupu:

```
RESERVATION      FIRST_NAME    LAST_NAME      NUM_OF_FLIGHTS  COST_ALL
----------- ---------------  ----------  ------------------  --------
          4            Ifor       Smoak                   1       512
          2         Sherwin         Hsu                   1       512
          5          Andrej        Nano                   2       624
          6          Teódor    Ladislav                   2       314
          2          Andrej        Nano                   1       410
          4           Peter       Marko                   1       123
          4          Teódor    Ladislav                   1       142
```

Po zavolaní príkazu EXPLAIN PLAN pre daný príkaz a vypísaní výstupu je možné vidieť, že optimalizátor Oracle vykonal postupne operácie SELECT, GROUP BY, JOIN a následne pristúpil k potrebným tabuľkám kompletným prechodom tabuľky. Taktiež je vidno výkonnostnú cenu jednotlivých operácií.

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |     9 |   954 |     7  (15)| 00:00:01 |
|   1 |  HASH GROUP BY      |            |     9 |   954 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN         |            |     9 |   954 |     6   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| TICKETS    |     9 |   351 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| PASSENGERS |    11 |   737 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

INDEX

Použitie indexovania môže byť užitočné v prípade častého vyhľadávania v určitej tabuľke. Naopak v prípade, že tabuľku často upravujeme, môže indexovanie prístup spomaliť, keďže je potrebné aktualizovať aj indexy.

V prípade spomenutého príkazu často pristupujeme k tabuľkám **tickets** a **passengers**. Pri vytváraní indexu je dôležité, aby bol vytvorený pre stĺpce v tabuľke, ktoré sú v príkazoch vyhľadávania priamo použité.
Z toho dôvodu bol vytvorený index:

```
CREATE INDEX ticket_index ON tickets (passenger, reservation, cost);
```

Indexované sú stĺpce, ktoré sú použité vo výberovej podmienke WHERE a následne tie, ktoré sú použité v príkaze SELECT.

Ak využijeme EXPLAIN PLAIN po druhý krát s aplikovaným indexom, je možné vidieť, že miesto prechodu celou tabuľkou bola zvolená operácia skenovania indexov. Táto zmena umožnila rýchlejšie/menej náročné prebehnutie väčšiny operácií v rámci príkazu.

```
----------------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |     9 |   954 |     5  (20)| 00:00:01 |
|   1 |  HASH GROUP BY      |              |     9 |   954 |     5  (20)| 00:00:01 |
|*  2 |   HASH JOIN         |              |     9 |   954 |     4   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN  | TICKET_INDEX |     9 |   351 |     1   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| PASSENGERS   |    11 |   737 |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------
```

Toto zlepšenie by bolo výrazne viditeľnejšie, ak by sme využívali tabuľku s väčším množstvom záznamov.

Ešte výraznejšie zrýchlenie vykonania príkazu by sa dalo dosiahnuť napríklad vytvorením druhého indexu:

```
CREATE INDEX passenger_index ON passengers (id, first_name, last_name);
```

Zrýchlenie môže taktiež nastať pri zmene typu spojenia. Napríklad LEFT OUTER JOIN je výrazne pomalší ako INNER JOIN. V našom príklade zmena ale nebola potrebná.

Prístupové práva

Prístupové práva by v kontexte našej témy mohli dávať zmysel napríklad pri týchto dvoch rolách zamestnancov pracujúcich s databázou:

a) Booking agent – človek zodpovedný za správu rezervácií, pasažierov, leteniek, zákaznikov, histórie vyhľadávania letov
b) Airline agent – človek zodpovedný za aktualizáciu dát o letoch, leteckých spoločnostiach a letiskách

V implementácii je ukážka pridania práv pre „Booking agenta“.

Materializovaný pohľad

Slúži na uloženie často využívaného pohľadu lokálne na disk, za účelom rýchleho prístupu pri opakovanom žiadaní o tento pohľad.

V našej implementácii sa nachádza materializovaný pohľad pre spoločné zobrazenie prepojenia cestujúcich s jednotlivými letenkami.

Sledujú sa zmeny na tabuľkách tickets a passengers.
Zmeny sa v danom pohľade aktualizuju až po vykonaní príkazu COMMIT.

V implementácii sa nachádza ukážka možnosti vyhľadávania v pohľade oproti inak nutnému vyhľadávaniu vo vzdialenej tabuľke. V prípade, ak po vykonaní príkazov upravujúcich vzdialenú tabuľku nie je zavolaný potvrdzovací príkaz COMMIT, nie sú zmeny do pohľadu aktualizované.