Bc. Pavol Loffay (učo 374115)
Studium FI N-AP SSME [sem 2, roč 1]
jaro 2014 (jiné)

Odpovědník Home Assignment No. 3

Odpovědi k průchodu Čt 15. 5. 2014 18:25.08, operaci Pá 16. 5. 2014 15:48.12, osobě P. Loffay, učo 374115

This assignment must be completed with meaningful answers when handing in.
Solutions with incomplete parts or filled with garbage will be automatically classified
as unsatisfactory without any later corrections possible! Work on the task individually.
You can work with the ROPOT repeatedly until the deadline, after which your answers
will be evaluated manually. There are some automatic value range test prepared just as
a pre-test for you -- if it fails, it may be a warning that you filled sth. in incorrectly.

• Klikněte: Ukaž Přehled nastavení parametrů odpovědníku.

Přehled nastavení parametrů odpovědníku

Kdy lze s odpovědníkem pracovat:

- od 6. 5. 2014 00:00 do 18. 5. 2014 23:59

Zobrazují se pouze správné odpovědi: ne
Test můžu skládat opakovaně: ano, pokaždé stejnou sadu otázek
Způsob bodování: neboduje se
Při vyplňování záleží na velikosti písmen: ne
Při vyplňování záleží na diakritice: ne
Při vyplňování nedovoluji zaměnit různé typy apostrofů a uvozovek: ne
Při vyplňování záleží na interpunkci: ne

Zeleně jsou vyznačeny správné odpovědi.

Query Optimization

In this task you will use the PostgreSQL database system and practise query optimization.

A simple guide for accessing the database can be found here. It is recommended to use the student DB server where all necessary tools are available. To access the faculty DB, follow the instructions from the first lecture and technical information at FI (in Czech only). You may use your own installation of PostgreSQL but make sure it is in the standard installation setting (in order to prevent customary settings from influencing your results).

The fundamental functionalities of the pgAdmin client are demonstrated in the video recordings of lectures "Query Tuning" and "Relational Schema Tuning" (both in Czech).

Additional documentation of the PosgreSQL DB can be found here, supported SQL commands are listed here and the documentation of functions and operators can be found here.

Creating a Table

After logging to the database (see the instructions above), execute the following commands that will create three tables (your own) and fill it with test data. The schema of relations are:

- *item(item_id, item_name)*
- *supplier(supplier_id, supplier_name)*
- *store(id, item_id, supplier_id, price, vat, quantity)*

Display the content of these relations and examine their content. The *price* attribute contains the price without VAT (value added tax), the *vat* attribute contains percentage of VAT and *quantity* the number of items available in stock.

```
CREATE TABLE item AS SELECT * FROM xdohnal.item;
CREATE TABLE supplier AS SELECT * FROM xdohnal.supplier;
CREATE TABLE store AS SELECT * FROM xdohnal.store;
```

1. Formulate an SQL query that retrieves the number of records in the relation *store* (0.5 point in total)

```
SELECT COUNT(*) FROM store;
```

SELECT COUNT(*) FROM store;

Enter the number of records returned: 250000 ✔ (250000)

Setting Primary Key

Set the attributes *id* in *store*, *item_id* in *item* and *supplier_id* in *supplier* to be primary keys.

2. Formulate the SQL query that returns the whole row of *store* whose id is the highest (maximum), i.e. the row that has been inserted into the relation *store* as last: (1.5 points in total)

```
SELECT * FROM store WHERE id=(SELECT MAX(id) FROM store);
```

SELECT * FROM store WHERE id=(SELECT MAX(id) FROM store);

Enter the execution plan for this query (command EXPLAIN):

```
 Index Scan using store_pkey on store  (cost=0.04..8.34 rows=1 width=31)
   Index Cond: (id = $1)
   InitPlan 2 (returns $1)
     ->  Result  (cost=0.03..0.04 rows=1 width=0)
           InitPlan 1 (returns $0)
             ->  Limit  (cost=0.00..0.03 rows=1 width=4)
                   ->  Index Scan Backward using store_pkey on store  (cost=0.00..8593.28 rows=250000 width=4)
                         Filter: (id IS NOT NULL)
(8 rows)
```

pgdb=> EXPLAIN SELECT * FROM store WHERE id=(SELECT MAX(id) FROM store); QUERY PLAN ---------- Index Scan using store_pkey on store (cost=0.04..8.34 rows=1 width=31) Index Cond: (id = $1) InitPlan 2 (returns $1) -> Result (cost=0.03..0.04 rows=1 width=0) InitPlan 1 (returns $0) -> Limit (cost=0.00..0.03 rows=1 width=4) -> Index Scan Backward using store_pkey on store (cost=0.00..8593.28 rows=250000 width=4) Filter: (id IS NOT NULL) (8 rows)

Enter the estimated query execution costs: 8.34 ✔ (0..9)

Enter the estimated result set size: 1 ✔ (1)rows

Describe briefly but accurately how the query will be evaluated by the database:

```
Pouzije sa index pre stlpec id.
Index je vytvoreny ako B-strom.
V poddotaze najde najvacsi prvok(id) a ten vrati.
Potom vo vonkajsom dotaze najde podla toho vrateneho id cely riadok a ten vrati.
```

Pouzije sa index pre stlpec id. Index je vytvoreny ako B-strom. V poddotaze najde najvacsi prvok(id) a ten vrati. Potom vo vonkajsom dotaze najde podla toho vrateneho id cely riadok a ten vrati.

Query Optimization

3. Formulate the query that retrieves supplier names, quantity and price for the item titled 'Handbag 1' that we have in stock and its price is at least 250 and up to 10 pieces available: (1 point in total)

```
SELECT supplier_name, quantity, price FROM store LEFT JOIN supplier AS sup ON
```

SELECT supplier_name, quantity, price FROM store LEFT JOIN supplier AS sup ON sup.supplier_id=store.supplier_id WHERE item_id=(select item_id from item where item_name='Handbag 1') AND price>= 250 AND quantity <= 10;

Enter the number of rows returned by the query: 3 ✔ (3)

Enter the execution plan of this query:

```
------------------------------------------------------------------------------
 Nested Loop Left Join  (cost=20.75..6488.83 rows=8 width=229)
   Join Filter: (sup.supplier_id = store.supplier_id)
   InitPlan 1 (returns $0)
     ->  Seq Scan on item  (cost=0.00..20.75 rows=1 width=8)
           Filter: ((item_name)::text = 'Handbag 1'::text)
   ->  Seq Scan on store  (cost=0.00..6459.00 rows=8 width=15)
         Filter: ((price >= 250::numeric) AND (quantity <= 10) AND (item_id = $0))
   ->  Seq Scan on supplier sup  (cost=0.00..1.06 rows=6 width=222)
(8 rows)
```

pgdb=> EXPLAIN SELECT supplier_name, quantity, price FROM store LEFT JOIN supplier AS sup ON sup.supplier_id=store.supplier_id WHERE item_id=(select item_id from item where item_name='Handbag 1') AND price>=250 AND quantity<=10; QUERY PLAN ----------

---------- Nested Loop Left Join (cost=20.75..6488.83 rows=8 width=229) Join Filter: (sup.supplier_id = store.supplier_id) InitPlan 1 (returns $0) -> Seq Scan on item (cost=0.00..20.75 rows=1 width=8) Filter: ((item_name)::text = 'Handbag 1'::text) -> Seq Scan on store (cost=0.00..6459.00 rows=8 width=15) Filter: ((price >= 250::numeric) AND (quantity <= 10) AND (item_id = $0)) -> Seq Scan on supplier sup (cost=0.00..1.06 rows=6 width=222) (8 rows)

Enter the estimated query execution costs: 6488.83 ✔ (6000..7000)

4. Optimize the previous query in such way that the execution will be significantly faster (at least 30% reduction in estimated costs).
Caveat: The optimization has to be generic, i.e. good for this type of query (so item name, price and quantity bounds may change)! (2 points in total)

Describe the idea of your optimization:

```
Skusal som preformulovat dotaz, aby sa selekcia robila cim najskor ale nikdy som nedostal vyrazne lepsiu
hodnotu ceny. Po uvahe som vytvoril index na tabulke store a stlcoch quantity a price.


CREATE INDEX index_store_quantity_price ON store (quantity,price);
```

Skusal som preformulovat dotaz, aby sa selekcia robila cim najskor ale nikdy som nedostal vyrazne lepsiu hodnotu ceny. Po uvahe som vytvoril index na tabulke store a stlcoch quantity a price. CREATE INDEX index_store_quantity_price ON store (quantity,price);

Enter the query execution plan after the optimization:

```
     ->  Seq Scan on item  (cost=0.00..20.75 rows=1 width=8)
           Filter: ((item_name)::text = 'Handbag 1'::text)
   ->  Bitmap Heap Scan on store  (cost=704.54..2949.00 rows=8 width=15)
         Recheck Cond: ((quantity <= 10) AND (price >= 250::numeric))
         Filter: (item_id = $0)
         ->  Bitmap Index Scan on nazev2  (cost=0.00..704.54 rows=9169 width=0)
               Index Cond: ((quantity <= 10) AND (price >= 250::numeric))
   ->  Hash  (cost=1.06..1.06 rows=6 width=222)
         ->  Seq Scan on supplier sup  (cost=0.00..1.06 rows=6 width=222)
(12 rows)
```

pgdb=> EXPLAIN SELECT supplier_name, quantity, price FROM store LEFT JOIN supplier AS sup ON sup.supplier_id=store.supplier_id WHERE item_id=(select item_id from item where item_name='Handbag 1') AND price>= 250 AND quantity <= 10; QUERY PLAN ---------- Hash Left Join (cost=726.43..2970.99 rows=8 width=229) Hash Cond: (store.supplier_id = sup.supplier_id) InitPlan 1 (returns $0) -> Seq Scan on item (cost=0.00..20.75 rows=1 width=8) Filter: ((item_name)::text = 'Handbag 1'::text) -> Bitmap Heap Scan on store (cost=704.54..2949.00 rows=8 width=15) Recheck Cond: ((quantity <= 10) AND (price >= 250::numeric)) Filter: (item_id = $0) -> Bitmap Index Scan on nazev2 (cost=0.00..704.54 rows=9169 width=0) Index Cond: ((quantity <= 10) AND (price >= 250::numeric)) -> Hash (cost=1.06..1.06 rows=6 width=222) -> Seq Scan on supplier sup (cost=0.00..1.06 rows=6 width=222) (12 rows)

Enter the final costs estimate: 2970.99 ✔ (5..4700)

- Zpět na výběr operace

Bez uložení:

- Zpět na výběr odpovědníku
- Moje studium
- Osobní administrativa