Carnegie Mellon University

15-415 Database Applications

Spring 2012, Faloutsos

Assignment 5: Query Optimization

Due: 3/20, 1:30 pm, in class – **hard copy**

Solution

Question 1: Query Optimization

[Q1.1]
Query Plan:

```
Seq Scan on play_in2  (cost=0.00..1446.65 rows=2841 width=26)
  Filter: (cast_position = 1)
```

[Q1.2]
Estimated cost: 1446.65

[Q1.3]
Query Plan:

```
Bitmap Heap Scan on play_in2  (cost=50.27..597.79 rows=2841 width=26)
  Recheck Cond: (cast_position = 1)
  ->  Bitmap Index Scan on cast_position_idx  (cost=0.00..49.56 rows=2841 width=0)
        Index Cond: (cast_position = 1)
```

[Q1.4]
Estimated cost: 597.79

[Q1.5]
The addition of the index made the query execution faster since the sequential scan is replaced by the index scan.

Question 2: Query Optimization 2

[Q2.1]
Query Plan:
Seq Scan on play_in2 (cost=0.00..1446.65 rows=118 width=26)
  Filter: ((name)::text ~~ '%smith%'::text)

[Q2.2]
Estimated cost: 1446.65

[Q2.3]
Query Plan:
Seq Scan on play_in2 (cost=0.00..1446.65 rows=118 width=26)
  Filter: ((name)::text ~~ '%smith%'::text)

[Q2.4]
Estimated cost: 1446.65

[Q2.5]
The addition of the index didn't change the query execution plan since the index doesn't help for the like query.

Question 3: Query Optimization 3

[Q3.1]
Query Plan:
Seq Scan on movies (cost=0.00..62.20 rows=893 width=36)
  Filter: ((rating * 3::double precision) > 20::double precision)

[Q3.2]
Estimated cost: 62.2

[Q3.3]
Query Plan:
Seq Scan on movies (cost=0.00..62.20 rows=893 width=36)
  Filter: ((rating * 3::double precision) > 20::double precision)

[Q3.4]
Estimated cost: 62.2

[Q3.5]
The addition of the index didn't change the query execution plan. There are two reasons:

1. The index on *rating* will not work for the query (*rating* * 3 > 20). Theoretically it might work in this case, but obviously the system is not "smart" enough to work that way; But you can create an index on (rating * 3) which will help.

2. You will notice that if the query is (*rating* > 6.667), PostgreSQL will still use sequential scan, since the cost to use index is even higher than sequential search. However, if the number of satisfying entries is very small (e.g., rating > 9), using indexing would be much faster, and the system will adopt it.

Question 4: Query Optimization 4

[Q4.1]
Query Plan:
Hash Join (cost=82.30..100471.32 rows=7641730 width=62) (actual time=3.702..7216.853 rows=8706104 loops=1)
Hash Cond: (play_in2.year = movies.year)
-> Seq Scan on play_in2 (cost=0.00..1259.72 rows=74772 width=26) (actual time=0.005..47.295 rows=74772 loops=1)
-> Hash (cost=48.80..48.80 rows=2680 width=36) (actual time=3.680..3.680 rows=2680 loops=1)
-> Seq Scan on movies (cost=0.00..48.80 rows=2680 width=36) (actual time=0.004..1.653 rows=2680 loops=1)

Estimated cost: 101773.42

[Q4.2]
Query Plan:
Nested Loop (cost=0.00..94120.93 rows=7641730 width=62) (actual time=0.046..17579.826 rows=8706104 loops=1)
-> Seq Scan on play_in2 (cost=0.00..1259.72 rows=74772 width=26) (actual time=0.004..48.191 rows=74772 loops=1)
-> Index Scan using movies_year_idx on movies (cost=0.00..0.83 rows=33 width=36) (actual time=0.004..0.089 rows=116 loops=74772)
Index Cond: (movies.year = play_in2.year)

Estimated cost: 94120.93

[Q4.3]
The addition of the index changed the query execution plan from the hsah join to indexed nested loop join, and thus the cost decreased.

[Q4.4]
Query Plan:

Nested Loop (cost=0.00..86903.80 rows=7641730 width=62) (actual time=0.044..16760.155 rows=8706104 loops=1)
-> Seq Scan on movies (cost=0.00..48.80 rows=2680 width=36) (actual time=0.005..1.779 rows=2680 loops=1)
-> Index Scan using play_in2_year_idx on play_in2 (cost=0.00..19.43 rows=1038 width=26) (actual time=0.008..2.318 rows=3249 loops=2680)
Index Cond: (play_in2.year = movies.year)

Estimated cost: 86903.80

[Q4.5]
The addition of the new index changed the query execution plan. The plan at Q4.2 used the nested loop with the index scan using movies_year_idx, while the plan at Q4.4 used the nested loop with the index scan using play_in2_year_idx, which even decreased the cost since the number of entries in the table movies is much smaller than the table play_in2.

Question 5: Query Optimization 5

[Q5.1]
Query Plan:
Hash Join (cost=82.30..2557.07 rows=74772 width=62)
Hash Cond: (play_in2.mid = movies.mid)
-> Seq Scan on play_in2 (cost=0.00..1259.72 rows=74772 width=26)
-> Hash (cost=48.80..48.80 rows=2680 width=36)
-> Seq Scan on movies (cost=0.00..48.80 rows=2680 width=36)

Estimated Cost: 2557.07

Join Algorithm: Hash Join

[Q5.2]
Disable hash join by using the “set enable_hashjoin=false;” command.

Query Plan:
Merge Join (cost=0.00..4662.33 rows=74772 width=62)
Merge Cond: (movies.mid = play_in2.mid)
-> Index Scan using movies_pkey on movies (cost=0.00..97.45 rows=2680 width=36)
-> Index Scan using play_in2_pkey on play_in2 (cost=0.00..3623.53 rows=74772 width=26)

Estimated Cost: 4662.33

Join Algorithm: Merge Join

[Q5.3]
Disable merge join by using the “set enable_mergejoin=false;” command.

Query Plan:
Nested Loop (cost=0.00..6851.67 rows=74772 width=62)
  -> Seq Scan on movies (cost=0.00..48.80 rows=2680 width=36)
  -> Index Scan using play_in2_pkey on play_in2 (cost=0.00..2.11 rows=34 width=26)
      Index Cond: (play_in2.mid = movies.mid)

Estimated Cost: 6851.67

Join Algorithm: Nested Loop Join