CENG352 Written Assignment 2 Solutions

Seda Civelek-2237147

May 2021

1 Query Processing

1.1 Part 1

a.

The query is trying to retrieve that, the age and maximum net worth of the actors that are in that age, provided that the ages are less than or equal to 30 and there are at least 3 actors in the same age. The minimal search key for B+tree index is (age,net_worth)

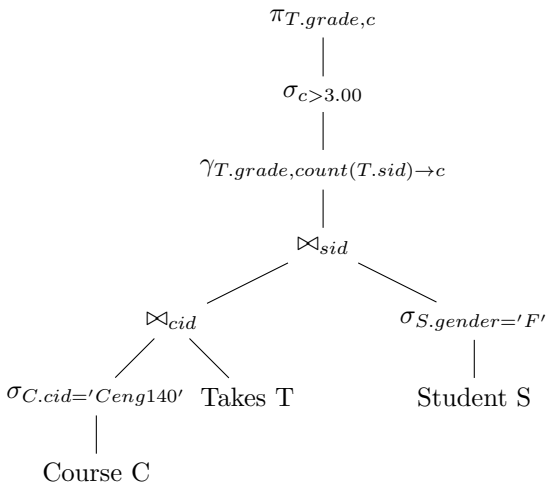
b.

i. This query can be evaluated with index-only plan using index on (year,budget) since in the query only year and budget attributes are used.

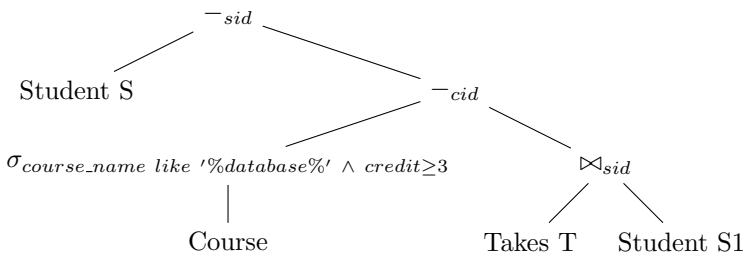
ii. This query can't be evaluated with index-only plan using index on (year,budget) since in the query we should retrieve the actual record for the genre information.

1.2 Part 2

a.



b.



2 Join Algorithms

$$T(R) = 40000$$

$$B(R) = 40000/10 = 4000$$

$$T(S) = 8000$$

$$B(S) = 8000/15 = 534$$

$$V(S, y) = 8000$$

$$M = 62$$

a.

$$\text{Cost} = B(R) + \frac{B(R).B(S)}{M-2}$$

$$\text{Cost} = 4000 + \frac{4000 * 534}{60} = 39600$$

b.

$$\text{Cost} = B(S) + \frac{B(S).B(R)}{M-2}$$

$$\text{Cost} = 534 + \frac{534 * 4000}{60} = 36134$$

c.

Check that;

$$B(R) \leq M^2$$
$$B(S) \leq M^2$$
$$4000 \leq 3844 \text{ (doesn't hold)}$$
$$534 \leq 3844$$

So, we can't apply two pass sort-merge join algorithm. We need more passes. General cost formula of sort-merge join is:

$$2B(R).ceil(\log_M(B(R))) + 2B(S).ceil(\log_M(B(S))) - B(R) - B(S)$$

$\text{Cost} = 2 * B(R) * ceil(\log_M(4000)) + 2 * B(S) * ceil(\log_M(534)) - B(R) - B(S)$
$\text{Cost} = 5B(R) + 3B(S) = 5 * 4000 + 3 * 534 = 21602$
All memory space is used for sort and merge operation.

d.

To be able to perform hash join, we must check that;

$$min(B(R), B(S)) \leq M^2$$
$$min(4000, 534) \leq 3844$$

Condition holds.

Step1: Hash R into M-1 buckets

Read R from disk to memory, Memory Space: 61 for R, 1 for input buffer $\rightarrow$ B(R)
Hash in memory
Write back R to disk $\rightarrow$ B(R)

Step2: Hash S into M-1 buckets

Read S from disk to memory, Memory space: 61 for S, 1 for input buffer $\rightarrow$ B(S)
Hash in memory
Write back S to disk $\rightarrow$ B(S)

Step3: Join Buckets

Read R from disk $\rightarrow$ B(R)
Read S from disk $\rightarrow$ B(S)
Memory space: 1 for input buffer, 1 for output buffer, 60 for hash table partition

Total Cost

Total cost = 3B(R)+3B(S)
Total cost = $3 * 4000 + 3 * 534 = 13602$

e.

For both clustered and unclustered index, we can traverse in B+tree index, find corresponding leaf nodes and retrieve actual data. In clustered index, we can retrieve similar index with getting one page. On the other hand, in unclustered index we may get more cache miss.
Index nested loop join steps are, iterate over blocks of R and for each block fetch corresponding block from S.

i. Cost = $B(R) + \frac{T(R) * B(S)}{V(S, y)}$
   Since y is a primary key of relation S, we must retrieve all records of S. So, $\frac{B(S)}{V(S,y)}$ ratio is 1.
   Cost = $4000 + 40000 * 1 = 44000$

ii. Cost = $B(R) + \frac{T(R) * T(S)}{V(S, y)}$
   Cost = $4000 + \frac{40000 * 8000}{8000} = 44000$

f.

i. Cost = $\frac{T(S)}{V(S, z)} * \frac{1}{\#tuplesperpage} = \frac{8000}{20} * \frac{1}{15} = 27$

ii. Cost = $\frac{T(S)}{V(S, z)} = \frac{8000}{20} = 400$

iii. Cost = $B(S) = 534$, it will make sequential search.

3 Physical Query Plan

a. Index scan:

Cost = $\frac{T(T)}{V(T, hashtag)} = \frac{600000}{50000} = 12$ I/O, because index is unclustered
Result size = $\frac{T(T)}{V(T, hashtag)} = \frac{600000}{50000} = 12$ tuple

b. Clustered index join:

Cost = $B(T) + \frac{T(T) * B(R)}{V(R, hashtag)} = 4000 + \frac{600000 * 90000}{30000} = 1804000$
Result size = $T(R) * T(T_{hashtag}) * selectivity = 3000000 * 12 * 1/10 = 3600000$

c. On-the-fly selection:

Cost = 0, since operation is done in memory
Result size = $\frac{T(R)}{V(R, user)} = \frac{3000000}{1000} = 3000$

d. Index scan:

Cost = $\frac{T(R)}{V(R, user)} = \frac{3000000}{1000} = 3000$ I/O, because index is unclustered
Result size = $\frac{T(R)}{V(R, user)} = \frac{3000000}{1000} = 3000$ tuple

e. In memory hash join, pipelined

In pipelined hash join, there are no read or write cost since it passes tuples directly to join.
Cost = 0
Result size = $T(Join) = T(T_{hashtag}) * T(R_{user}) * selectivity = 12 * 3000 * 1/10 =$ 3600 (Selectivity factor is unknown, so take it 1/10

Total cost:

Plan1 cost = $12 + 1804000 = 1804012$
Plan2 cost = $12 + 3000 = 3012$
I would choose plan2 since we increase the total number of tuples to be joined before the join operation, and pipelined hash join doesn't use read and write

operations from disk.

4 Experiments

a.

I guessed that dbms chooses hash-join method, because other methods are not appropriate for this kind of query since there is equality check. Command to check execution time:

```
explain analyze
select *
from business
natural join tip;
```

	QUERY PLAN
1	Hash Join (cost=7746.16..90913.99 rows=1162119 width=176) (actual time=47.209..1101.483 rows=1162119 loops=1)
2	Hash Cond: ((tip.business_id)::text = (business.business_id)::text)
3	-> Seq Scan on tip (cost=0.00..34952.19 rows=1162119 width=125) (actual time=0.013..218.889 rows=1162119 loops=1)
4	-> Hash (cost=3699.85..3699.85 rows=160585 width=74) (actual time=47.016..47.016 rows=160585 loops=1)
5	Buckets: 65536 Batches: 8 Memory Usage: 2665kB
6	-> Seq Scan on business (cost=0.00..3699.85 rows=160585 width=74) (actual time=0.006..8.752 rows=160585 loops=1)
7	Planning Time: 1.871 ms
8	Execution Time: 1123.236 ms

Figure 1: Execution time before indexing

b.

I created 2 indexes both on business and tip. Command to create indexes:

```
create index t_bidx on tip using hash(business_id);
create index b_bidx on business using btree(business_id);
```

Before creating indexes, execution time was 1123ms. After creating indexes on business_id execution time decreased to 1118ms. I created indexes as hash and btree. I found them by trial and error method.

	QUERY PLAN
1	Hash Join (cost=7746.16..90913.99 rows=1162119 width=176) (actual time=47.828..1097.058 rows=1162119 loops=1)
2	Hash Cond: ((tip.business_id)::text = (business.business_id)::text)
3	-> Seq Scan on tip (cost=0.00..34952.19 rows=1162119 width=125) (actual time=0.008..221.828 rows=1162119 loops
4	-> Hash (cost=3699.85..3699.85 rows=160585 width=74) (actual time=46.370..46.371 rows=160585 loops=1)
5	Buckets: 65536 Batches: 8 Memory Usage: 2665kB
6	-> Seq Scan on business (cost=0.00..3699.85 rows=160585 width=74) (actual time=0.006..7.487 rows=160585 loo
7	Planning Time: 3.506 ms
8	Execution Time: 1118.923 ms

Figure 2: Execution time after indexing

c.

I guessed that the dbms chooses nested-loop join method, because it uses comparison operation instead of equality check. Command to see execution plan:

```
explain
select *
from tip
inner join business on  business.business_id < tip.business_id;
```

	QUERY PLAN
1	Nested Loop (cost=0.42..1711197028.55 rows=62206293205 width=199)
2	-> Seq Scan on tip (cost=0.00..34952.19 rows=1162119 width=125)
3	-> Index Scan using b_bidx on business (cost=0.42..937.17 rows=53528
4	Index Cond: ((business_id)::text < (tip.business_id)::text)

Figure 3: Execution plan of join where Business.business id < Tip.business id.

d.

When I disable hash join, DBMS chooses merge join method if the indexes are dropped. If the indexes are still visible, it chooses nested loop. Command to disable method:

```
set enable_hashjoin=off;
```

When I disable nested loop join, DBMS continue to choose nested loop method both for indexed tables and no-indexed tables. Command to disable method:

```
set enable_nestloop=off;
```

e.

Whenever I create the index with following command:

```
create index state_star_idx on business (state, stars);
cluster state_star_idx on business;
```

I created index on state and stars on table Business. Then, I turned the index as clustered.

f.

Yes, it will affect the execution plans of other queries since by constructing clustered index on the table, we changed the places of records.