1 Query Optimization

Consider a relation with this schema:

Employees(*eid* : **integer**, *ename* : **string**, *sal* : **integer**, *title* : **string**, *age* : **integer**)

Suppose that the following indexes, all using indirect storage (i.e., an index entry is a pointer to the record where the data is) for data entries, exist: a hash index on *eid*, a B+ tree index on *sal*, a hash index on *age*, and a clustered B+-tree index on $<age, sal>$. Each Employees record is 100 bytes long, and you can assume that each index data entry is 20 bytes long. The Employees relation contains 10,000 pages.

1. Consider each of the following selection conditions and, assuming that the reduction factor (RF) for each term that matches an index is 0.1, compute the cost of the most selective access path for retrieving all Employees tuples that satisfy the condition:

   (a) $sal > 100$
   (b) $age = 25$
   (c) $age > 20$
   (d) $eid = 1,000$
   (e) $sal > 200 \wedge age > 30$
   (f) $sal > 200 \wedge age = 20$
   (g) $sal > 200 \wedge title =' CFO'$
   (h) $sal > 200 \wedge age > 30 \wedge title =' CFO'$

2. Suppose that, for each of the preceding selection conditions, you want to retrieve the average salary of qualifying tuples. For each selection condition, describe the least expensive evaluation method and state its cost.

3. Suppose that, for each of the preceding selection conditions, you want to compute the average salary for each age group. For each selection condition, describe the least expensive evaluation method and state its cost.

4. Suppose that, for each of the preceding selection conditions, you want to compute the average age for each *sal* level (i.e., group by *sal*). For each selection condition, describe the least expensive evaluation method and state its cost.

5. For each of the following selection conditions, describe the best evaluation method:

   (a) $sal > 200 \vee age = 20$
   (b) $sal > 200 \vee title =' CFO'$
   (c) $title =' CFO' \vee ename =' Joe'$

Solution

1. For this problem, it will be assumed that each data page contains 20 relations per page.

   (a) $sal > 100$: For this condition, a filescan would probably be best, since a clustered index does not exist on *sal*. Using the unclustered index would result in a cost of 10,000 pages * $\frac{20bytes}{100bytes}$ * 0.1 for the B+ index scan plus 10,000 pages * 20 tuples per page * 0.1 for the lookup = 22000, and would be inferior to the filescan cost of 10000.

(b) $age = 25$: The clustered B+-tree index would be the best option here, with a cost of 2 (lookup) + 10000 pages * 0.1 (selectivity) + 10,000 * 0.2 (reduction)* 0.1 = 1202. Although the hash index has a lesser lookup time, the potential number of record lookups (10000 pages * 0.1 * 20 tuples per page = 20000) renders the clustered index more efficient.

(c) $age > 20$: Again the clustered B+-tree index is the best of the options presented; the cost of this is 2 (lookup) + 10000 pages * 0.1 (selectivity)+ 200 = 1202.

(d) $eid = 1,000$: Since *eid* is a candidate key, one can assume that only one record will be in each bucket. Thus, the total cost is roughly 1.2 (lookup) + 1 (record access) which is 2 or 3.

(e) $sal > 200 \wedge age > 30$: This query is similar to the $age > 20$ case if the $age > 30$ clause is examined first. Then, the cost is again 1202.

(f) $sal > 200 \wedge age = 20$: Similar to the previous part, the cost for this case using the clustered B+ index on $<age, sal>$ is smaller, since only 10% of all relations fulfill $sal > 200$. Assuming a linear distribution of values for *sal* for age, one can assume a cost of 2 (lookup) + 10000 pages * 0.1 (selectivity for *age*) * 0.1 (selectivity for *sal*) + 10,000 * 0.4 * 0.1 * 0.1 = 142.

(g) $sal > 200 \wedge title =' CFO'$: In this case, the filescan is the best available method to use, with a cost of 10000.

(h) $sal > 200 \wedge age > 30 \wedge title =' CFO'$: Here an age condition is present, so the clustered B+ tree index on $<age, sal>$ can be used. Here, the cost is 2 (lookup) + 10000 pages * 0.1 (selectivity) = 1002.

2. **(a)** $sal > 100$: Since the result desired is only the average salary, an index-only scan can be performed using the unclusterd B+-tree on *sal* for a cost of 2 (lookup) + 10000 * 0.1 * 0.2 (due to smaller index tuples) = 202.

   (b) $age = 25$: For this case, the best option is to use the clustered index on $<age, sal>$, since it will avoid a relational lookup. The cost of this operation is 2 (B+-tree lookup) + 10000 * 0.1 * 0.4 (due to smaller index tuple sizes) = 402.

   (c) $age > 20$: Similar to the $age = 25$ case, this will cost 402 using the clustered index.

   (d) $eid = 1,000$: Being a candidate key, only one relation matching this should exist. Thus, using the hash index again is the best option, for a cost of 1.2 (hash lookup) + 1 (relation retrieval) = 2.2.

   (e) $sal > 200 \wedge age > 30$: Using the clustered B+-tree again as above is the best option, with a cost of 402.

   (f) $sal > 200 \wedge age = 20$: Similarly to the $sal > 200 \wedge age = 20$ case in the previous problem, this selection should use the clustered B+ index for an index only scan, costing 2 (B+ lookup) + 10000 * 0.1 (selectivity for *age*) * 0.1 (selectivity for *sal*) * 0.4 (smaller tuple sizes, index-only scan) = 42.

   (g) $sal > 200 \wedge title =' CFO'$: In this case, an index-only scan may not be used, and individual relations must be retrieved from the data pages. The cheapest method available is a simple filescan, with a cost of 10000 I/Os.

   (h) $sal > 200 \wedge age > 30 \wedge title =' CFO'$: Since this query includes an age restriction, the clustered B+ index over $<age, sal>$ can be used; however, the inclusion of the title field precludes an index-only query. Thus, the cost will be 2 (B+-tree lookup) + 10000 * 0.1 (selectivity on *age*) + 10,000 * 0.1 * 0.4 = 1402 I/Os.

3. **(a)** $sal > 100$: The best method in terms of I/O cost requires usage of the clustered B+ index over $<age, sal>$ in an index-only scan. Also, this assumes the ablility to keep a running average for each age category. The total cost of this plan is 2 (lookup on B+-tree, find min entry) + 10000 * 0.4 (index-only scan) = 4002. Note that although *sal* is part of the key, since it is not a $prefix$ of the key, the entire list of pages must be scanned.

(b) $age = 25$: Again, the best method is to use the clustered B+ index in an index-only scan. For this selection condition, this will cost 2 (age lookup in B+-tree) + 10000 pages * 0.1 (selectivity on age) * 0.4 (index-only scan, smaller tuples, more per page, etc.) = 2 + 400 = 402.

(c) $age > 20$: This selection uses the same method as the previous condition, the clustered B+-tree index over $<age, sal>$ in an index-only scan, for a total cost of 402.

(d) $eid = 1,000$: As in previous questions, eid is a candidate field, and as such should have only one match for each equality condition. Thus, the hash index over eid should be the most cost effective method for selecting over this condition, costing 1.2 (hash lookup) + 1 (relation retrieval) = 2.2.

(e) $sal > 200 \wedge age > 30$: This can be done with the clustered B+ index and an index-only scan over the $<age, sal>$ fields. The total estimated cost is 2 (B+ lookup) + 10000 pages * 0.1 (selectivity on age) * 0.4 (index-only scan) = 402.

(f) $sal > 200 \wedge age = 20$: This is similar to the previous selection conditions, but even cheaper. Using the same index-only scan as before (the clustered B+ index over $<age, sal>$), the cost should be 2 + 10000 * 0.4 * 0.1 (age selectivity) * 0.1 (sal selectivity) = 42.

(g) $sal > 200 \wedge title =' CFO'$: Since the results must be grouped by age, a scan of the clustered $<age, sal>$ index, getting each result from the relation pages, should be the cheapest. This should cost 2 + 10000 * 0.4 + 10000 * tuples per page * 0.1 + 5000 * 0.1 (index scan cost) = 2 + 1000 * (4 + tuples per page). Assuming the previous number of tuples per page (20), the total cost would be 24002. Sorting the filescan alone, would cost 40000 I/Os. However, if the tuples per page is greater than 36, then sorting the filescan would be the best, with a cost of 40000 + 6000 (secondary scan, with the assumption that unneeded attributes of the relation have been discarded).

(h) $sal > 200 \wedge age > 30 \wedge title =' CFO'$: Using the clustered B+-tree over $<age, sal>$ would result in a cost of 2 + 10000 * 0.1 (selectivity of age) + 5000 * 0.1 = 1502 lookups.

4. **(a)** $sal > 100$: The best operation involves an external merge sort over $<sal, age>$, discarding unimportant attributes, followed by a binary search to locate minimum $sal < 100$ and a scan of the remainder of the sort. This costs a total of 16000 (sort) + 12 (binary search) + 10000 * 0.4 (smaller tuples) * 0.1 (selectivity of sal) + 2 = 16000 + 4000 + 12 + 400 + 2= 16414.

(b) $age = 25$: The most cost effective technique here employs sorting the clustered B+ index over $<age, sal>$, as the grouping requires that the output be sorted. An external merge sort with 11 buffer pages would require 16000. Totalled, the cost equals 16000 (sort) + 10000 * 0.4 = 20000.

(c) $age > 20$: This selection criterion works similarly to the previous one, in that an external merge over $<age, sal >$ is required, using the clustered index provided as the pages to sort. The final cost is the same, 20000.

(d) $eid = 1,000$: Being a candidate key, only one relation should match with a given eid value. Thus, the estimated cost should be 1.2 (hash lookup) + 1 (relation retrieval).

(e) $sal > 200 \wedge age > 30$: This case is similar to the $sal > 100$ case above, cost = 16412.

(f) $sal > 200 \wedge age = 20$: Again, this case is also similar to the $sal > 100$ case, cost = 16412.

(g) $sal > 200 \wedge title =' CFO'$: The solution to this case greatly depends of the number of tuples per page. Assuming a small number of tuples per page, the cheapest route is to use the B+-tree index over sal, getting each index. The total cost for this is 2 (lookup, $sal > 200$) + 10000 * 0.2 (smaller size) * 0.1 (selectivity) + 10000 * 0.1 (selectivity) * tuples per page. The solution to this case is similar to that of the other requiring sorts, but at a higher cost. Since the sort can't be preformed over the clustered B+-tree in this case, the sort costs 40000 I/Os. Thus, for tuples per page ≤ 40, the B+ index method is superior, otherwise, the sort solution is cheaper.

(h) $sal > 200 \wedge age > 30 \wedge title =' CFO'$: This solution is the same as the previous, since either the index over *sal* or an external sort must be used. The cost is the cheaper of 2 + 1000 * (.2 + tuples per page) [index method] and 40000 [sort method].

5. **(a)** $sal > 200 \vee age = 20$: In this case, a filescan would be the most cost effective, because the most cost effective method for satisfying $sal > 200$ alone is a filescan.

   (b) $sal > 200 \vee title =' CFO'$: Again a filescan is the better alternative here, since no index at all exists for *title*.

   (c) $title =' CFO' \vee ename =' Joe'$: Even though this condition is a conjunction, the filescan is still the best method, since no indexes exist on either *title* or *ename*.

2 Query Optimization

Consider the following relational schema and SQL query. The schema captures information about employees, departments, and company finances (organized on a per department basis).

Emp(*eid* : **integer**, *did* : **integer**, *sal* : **integer**, *hobby* : **char(20)**)

Dept(*did* : **integer**, *dname* : **char(20)**, *floor* : **integer**, *phone* : **char(10)**)

Finance(*did* : **integer**, *budget* : **real**, *sales* : **real**, *expenses* : **real**)

Consider the following query:

```
SELECT D.dname, F.budget
FROM Emp E, Dept D, Finance F
WHERE E.did=D.did AND D.did=F.did AND D.floor=1
AND E.sal ≥ 59000 AND E.hobby = 'yodeling'
```

1. Identify a relational algebra tree (or a relational algebra expression if you prefer) that reflects the order of operations a decent query optimizer would choose.

2. List the join orders (i.e., orders in which pairs of relations can be joined to compute the query result) that a relational query optimizer will consider. (Assume that the optimizer follows the heuristic of never considering plans that require the computation of cross-products.) Briefly explain how you arrived at your list.

3. Suppose that the following additional information is available: Unclustered B+ tree indexes exist on *Emp.did*, *Emp.sal*, *Dept.floor*, *Dept.did*, and *Finance.did*. The system's statistics indicate that employee salaries range from 10,000 to 60,000, employees enjoy 200 different hobbies, and the company owns two floors in the building. There are a total of 50,000 employees and 5,000 departments (each with corresponding financial information) in the database. The DBMS used by the company has just one join method available, index nested loops.

   (a) For each of the query's base relations (Emp, Dept, and Finance) estimate the number of tuples that would be initially selected from that relation if all of the non-join predicates on that relation were applied to it before any join processing begins.

   (b) Given your answer to the preceding question, which of the join orders considered by the optimizer has the lowest estimated cost?

Solution

1. $$\pi_{D.dname,F.budget}((\pi_{E.did}(\sigma_{E.sal\geq 59000,E.hobby=\text{“}yodelling\text{”}}(E)) \bowtie \pi_{D.did,D.dname}(\sigma_{D.floor=1}(D))) \bowtie \pi_{F.budget,F.did}(F))$$

2. There are 2 join orders considered, assuming that the optimizer only consider left-deep joins and ignores cross-products: (D,E,F) and (D,F,E)

3. **(a)** The answer to each relation is given below.
   - Emp: card = 50,000, E.sal >= 59,000, E.hobby = “yodelling”
     resulting card = 50000 * 1/50 * 1/200 = 5
   - Dept: card = 5000, D.floor = 1
     resulting card = 5000 * 1/2 = 2500
   - Finance: card = 5000, there are no non-join predicates
     resulting card = 5000

   (b) Consider the following join methods on the following left-deep tree: $((E \bowtie D) \bowtie F)$. The tuples from E will be pipelined, no temporary relations are created.

   First, retrieve the tuples from E with salary 0 59,000 using the B-tree index on salary; we estimate 1000 such tuples will be found, with a cost of 1 tree traversal + the cost of retrieving the 1000 tuples (since the index is unclustered) = 3+1000 = 1003. Note, we ignore the cost of scanning the leaves.

   Of these 1000 retrieved tuples, on the fly select only those that have hobby = “yodelling”, we estimate there will be 5 such tuples.

   Pipeline these 5 tuples one at a time to D, and using the B-tree index on D.did and the fact the D.did is a key, we can find the matching tuples for the join by searching the Btree and retrieving at most 1 matching tuple, for a total cost of 5(3 + 1) = 20. The resulting cardinality of this join is at most 5.

   Pipeline the estimated 3 tuples of these 5 that have D.floor=1 1 up to F, and use the Btree index on F.did and the fact that F.did is a key to retrieve at most 1 F tuple for each of the 3 pipelined tuples. This costs at most 3(3+1) = 12.

   Ignoring the cost of writing out the final result, we get a total cost of 1003+20+12 = 1035.

3 Concurrency Control

Read the article “Concurrency Control and Recovery” by Mike Franklin (you can find it at http://infosys.cs.uni-saarland.de/teaching/dbs0809/exercises/franklin.pdf).

1. Give examples of one serializable and one non-serializable schedule and create precedence graphs for them. Also point out the conflicting pairs.

2. Explain the differences between STEAL and NO-STEAL and between FORCE and NO-FORCE policies.

3. Explain the differences between the “basic” 2PL and strict 2PL.

4. What is the phantom problem? Given an example of how it could occur and how could it be avoided.

Solution

1. see lecture slides for week 10b
2. STEAL: dirty pages may be evicted from the DB-bzffer; NO-STEAL: dirty pages may not be evicted from the DB-buffer; FORCE: if a page was changed by a TA that page will be forced to disk when the TA commits; NO-FORCE: if a page was changed by a TA that page will not be forced to disk when the TA commits
3. strict 2PL: TA will not release any lock before end of TA (EOT)
4. What is the phantom problem: a TA may see different states of the database in the sense that a TA executes two queries based on the same prredicate twice. Both query exectuions may yield different results as a concurrent TA may have inserted additional rows into the DB. This violates isolation. The problem occurs in the weaker isolation level "repeatable read" due to the fact that the DBMS does not lock the different access paths to the data. To fix this problem you need to introduce predicate locks. This is done in isolation level "serializable".

All solutions to these questions available on the slides.

4 Strict 2PL

Show how the following transaction histories would be handled by a scheduler based on strict 2PL. Add information to the transaction histories regarding:

- which locks are acquired (and when);
- whether any deadlock is incurred at any point;
- what is the waits-for graph for each deadlock situation;
- which transactions are aborted in the event of a deadlock;
- which transactions succeed.

(a) R1(A) R2(A) W1(A) W2(A) C1 C2

(b) W3(A) R1(A) W1(Z) R2(B) W2(Y) W3(B) C1 C2 C3

(c) R1(A) R2(B) R3(A) W1(A) R3(B) W3(B) A2 R3(C) W3(C) C3 C1

Solution

(a) The revised history is shown below.

LS1(A) R1(A) LS2(A) R2(A) LX1(A) [here T1 blocks, as T2 has a shared lock on A]

LX2(A) [here T2 blocks, as T1 has a shared lock on A]

At this moment, there is a deadlock. The waits-for graph contains the edges T1 → T2 and T2 → T1. The scheduler must choose one transaction to be aborted, say T2.

A2 W1(A) C1

T2 may be now retried as transaction T3.

LS3(A) R3(A) LX3(A) W3(A) C3

(b) The revised history is shown below. No deadlock occurs in this history.

LX3(A) W3(A) LS1(A) [here T1 blocks, as T3 has an exclusive lock on A]

LS2(B) R2(B) LX2(Y) W2(Y) LX3(B) [here T3 blocks, as T2 has a shared lock on B]

C2 [T3 continues]

W3(B) C3 [T1 continues]

R1(A) LX1(Z) W1(Z) C1

(c) The revised history is shown below.

LS1(A) R1(A) LS2(B) R2(B) LS3(A) R3(A) LX1(A) [here T1 blocks, as T3 has a shared lock on A]

LS3(B) R3(B) LX3(B) [here T3 blocks, as T2 has a shared lock on B]

A2 [T3 continues]

W3(B) LS3(C) LX3(C) W3(C) C3 [T1 continues]

W1(A) C1