Question 1

Consider two relations A and B. A has 80,000 tuples, and B has 100,000 tuples. Both relations store 100 tuples per page. Consider the following SQL statement:

SELECT *

FROM A INNER JOIN B

ON A.a = B.a;

We wish to evaluate an equijoin between A and B, with an equality condition A.a = B.a. There are 102 buffer pages available for this operation. Both relations are stored as (unsorted) heap files. Neither relation has any indexes built on it.

Consider the alternative join strategies described below and calculate the cost of each alternative. Evaluate the algorithms using the number of disk I/O's (i.e., pages) as the cost. For each strategy, provide the formulae you use to calculate your cost estimates.

a) Page-oriented Nested Loops Join. Consider A as the outer relation.

(a) Page-oriented Nested Loops join

Formula: Cost(PNLJ) = NPages(Outer)+Npages(Outer)*Npages(Inner)

Npages(Outer) = 80000 / 100 = 800

Npages(Inner) = 100000/100 = 1000

Cost(PNLJ) = 800 + 800*1000 = 800800 I/O

b) Block-oriented Nested Loops Join. Consider A as the outer relation.

(b) Block-oriented Nested Loops join

Formula: Cost(BNLJ) = Npages(Outer)+NBlocks(Outer)*Npages(Inner)

Nblocks(Outer) = Npages(Outer) / (B-2) = 800 / (102-2) = 8

Cost(BNLJ) = 800 + 8*1000 = 8800 I/O

c) Sort-Merge Join. Assume that Sort-Merge Join can be done in 2 passes.

(c) Sort-Merge join with 2 passes

Formula: Sort(Outer)+Sort(Inner)+Npages(Outer)+Npages(Inner)

Sort(R) = 2*Numpass*Npages(R) = 2*2*Npages(R) = 4Npages(R)

Cost(SMJ) = 4*800+4*1000+800+1000 = 9000 I/O

d) Hash Join

(d) Hash join

Formula: Cost(HJ) = 2Npages(Outer)+2Npages(Inner)+Npages(Outer)+Npages(Inner)

Cost(HJ) = 2*800+2*1000+800+1000 = 5400 I/O

e) What would be the lowest possible cost to perform this query, assuming that no indexes are built on any of the two relations, and assuming that sufficient buffer space is available? What would be the minimum buffer size required to achieve this cost? Explain briefly.

(e) The lowest possible cost would happen if each relations are only read once to complete the join. Storing the smaller relation (A) in memory and read in thee larger relation (B) page by page and for each tuple in larger relation (B), we search the smaller relation (A) for matching tuples. The buffer pool would hold the entire smaller relation (A), one page for reading B and one page to serves as output buffer

Cost = 800 + 1000 = 1800 I/O

Minimum buffer size = 800 + 1 + 1 = 802

Question 2

Consider a relation with the following schema:

JobSeekers(id, firstname, lastname, city, soughtsalary)

The JobSeekers relation consists of 10,000 pages. Each page stores 100 tuples. The online software works in the 8 largest Australian cities and soughtsalary can have values between 60,000 and 160,000 (i.e., [60,000 – 160,000].)

Suppose that the following SQL query is executed frequently using the given relation:

SELECT *

FROM JobSeekers

WHERE city = 'Melbourne' AND soughtsalary > 80,000;

a) Compute the reduction factors and the estimated result size in number of tuples.

(a) As 8 cities, soughtsalary value is between 60000~160000, for selecting where
City = 'Melbourne' And soughtsalary > 80000;
Reduction Factor:
RF(city) = 1 / Nkeys(Col) = 1 / 8 = 12.5%
RF(soughtsalary) = (High(Col) – value) / (High(Col) – Low(Col))
RF(soughtsalary) = (160000-80000) / (160000-60000) = 80%
RF(city and soughtsalary) = RF(city) * RF(soughtsalary) = 12.5% * 80% = 10%
Result size:
RS(city) = 10000 * 100 * 12.5% = 125000 tuples
RS(soughtsalary) = 10000 * 100 * 80% = 800000 tuples
RS(city and soughtsalary) = 10000* 100 * 10% = 100000 tuples

b) Compute the estimated cost in number of disk I/O's of the best plan if a clustered B+ tree index on (city, soughtsalary) is the only index available. Suppose there are 2,000 index pages. Discuss and calculate alternative plans.

(b) Clustered B+ tree index on (city, soughtsalary)

RF = 10%, indexPages = 2000

Cost = (NPages(I) + NPages(R))*RF = (2000 + 10000) * 10% = 1200 I/O

Heap Scan Cost = Npages(R) = 10000 I/O

Cost matches index on city = 12.5%*(2000+10000) = 1500 I/O

10000 > 1500 > 1200

Therefore the cheapest access path is to use the B+ tree index with cost 1200 I/O

c) Compute the estimated cost in number of disk I/O's of the best plan if an unclustered B+ tree index on (soughtsalary) is the only index available. Suppose there are 2,000 index pages. Discuss and calculate alternative plans.

(c) Unclustered B+ tree index on (soughtsalary)

RF = 80%, indexPages = 2000

Cost = (NPages(I) + NTuples(R))*RF = (2000 + 10000*100) * 80% = 801600 I/O

Heap Scan Cost = Npages(R) = 10000 I/O

801600 > 10000

Therefore the cheapest access path is to use the full table scan with cost 10000 I/O

d) Compute the estimated cost in number of disk I/O's of the best plan if an unclustered Hash index on (city) is the only index available. Discuss and calculate alternative plans.

(d) Unclustered Hash index on (city)

RF = 12.5%, hash lookup cost = 2.2

Cost = RF*hash lookup cost*Ntuples(R) = 12.5% * 2.2 * 10000 * 100 = 275000 I/O

Heap Scan Cost = Npages(R) = 10000 I/O

275000 > 10000

Therefore the cheapest access path is to use the full table scan with cost 10000 I/O

e) Compute the estimated cost in number of disk I/O's of the best plan if an unclustered Hash index on (soughtsalary) is the only index available. Discuss and calculate alternative plans.

(e) Unclustered Hash index on (soughtsalary)

RF = 80%, hash lookup cost = 2.2|

Cost = RF*hash lookup cost*Ntuples(R) = 80% * 2.2 * 10000 * 100 = 1760000 I/O

Heap Scan Cost = Npages(R) = 10000 I/O

1760000 > 10000

Therefore the cheapest access path is to use the full table scan with cost 10000 I/O

Question 3

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 E.sal > 100,000

AND E.hobby IN (‘diving’, ‘soccer’);

The system’s statistics indicate that employee salaries range from 50,000 to 150,000, and employees enjoy 50 different hobbies. There is a total of 25,000 employees and 1,200 departments (each with corresponding financial record in the Finance relation) in the database. Each relation fits 100 tuples in a page. Suppose there exists a clustered B+ tree index on (Dept.did) and a clustered B+ tree index on (Emp.salary), both of size 50 pages.

a) Compute the reduction factors and the estimated result size in number of tuples.

(a) Reduction Factor

RF(E.did = D.did) = Max (NKeys(E.did), NKeys(D.did)) = 1/1200

RF(D.did = F.did) = Max (NKeys(D.did), NKeys(F.did)) = 1/1200

RF(E.sal > 100000) = (High(Col) – value) / (High(Col) – Low(Col))

= (150000 - 100000) / (150000 - 100000) = 1/2

RF(E.hobby IN (‘diving’, ‘soccer’)) = 1/50 + 1/50 = 2/50

RF(All Predicates) = 1/1200 * 1/1200 * 1/2 * 2/50 = 1/72000000

Result Size = RF(All Predicates) * NTuples(D) * NTuples(E) * NTuples(F)

= 1/72000000 * 1200 * 25000 * 1200 = 500 tuples

b) Compute the cost in number of disk I/O’s of the plans shown below. Assume that sorting of any relation (if required) can be done in 2 passes. NLJ is a Page-oriented Nested Loops Join. Assume that did is the candidate key, and that 50 tuples of a resulting join between Emp and Dept fit in a page. Similarly, 50 tuples of a resulting join between Finance and Dept fit in a page. Any selections / projections not indicated on the plan are performed “on the fly” after all joins have been completed.

b) 1) Cost(PNLJ) = NPages(Outer) + Npages(Outer) * Npages(Inner)

Cost(DxF) = 12 + 12 * 12 = 156 I/O

Result Size(DxF) = 12 * 100 * 12 * 100 / 1200 = 1200 tuples = 24 pages

Cost(xE) = 24 * 250 = 6000 I/O

Cost(Total) = 156 + 6000 = 6156 I/O

b) 2) Cost(HJ) = 2*NPages(Outer) + 2*NPages(Inner) + NPages(Outer) + NPages(Inner)

Cost(SMJ) = Sort(Outer) + Sort(Inner) + NPages(Outer) + NPages(Inner)

Sort(R) = 2*NumPasses*NPages(R), NumPasses = 2

Cost(DxF) = 3 * 12 + 3 * 12 = 72 I/O

Result Size(DxF) = 12 * 100 * 12 * 100 / 1200 = 1200 tuples = 24 pages

Cost(xE) = 24 * 4 + 5 * 250 = 1346 I/O

Cost(Total) = 72 + 1346 = 1418 I/O

b) 3) Cost(SMJ) = Sort(Outer) + Sort(Inner) + NPages(Outer) + NPages(Inner)

Sort(R) = 2*NumPasses*NPages(R), NumPasses = 2

Cost(HJ) = 2*NPages(Outer) + 2*NPages(Inner) + NPages(Outer) + NPages(Inner)

Cost(ExD) = 4 * 250 + 250 + 12 = 1262 I/O

Result Size(ExD) = 250 * 100 * 12 * 100 / 1200 = 25000 tuples = 500 pages

Cost(xF) = 2 * 500 + 3 * 12 = 1036 I/O

Cost(Total) = 1262 + 1036 = 2298 I/O

b) 4) Cost(HJ) = 2*NPages(Outer) + 2*NPages(Inner) + NPages(Outer) + NPages(Inner)

Cost(PNLJ) = NPages(Outer) + Npages(Outer) * Npages(Inner)

Cost(Selection) = (NPages(I) + NPages(R))*RF, RF = 1/2

Cost(Selection) = (50 + 250) / 2 = 150 I/O

Cost(SelectedExD) = 2 * 125 + 2 * 12 + 125 + 12 = 411 I/O

As doing selection has already store the Outer Npages, Cost(SelectedExD) = 411 - 125 = 286 I/O

Result Size(SelectedExD) = 125 * 100 * 12 * 100 / 1200 = 12500 tuples = 250 pages

Cost(xF) = 250 * 12 = 3000 I/O

Cost(Total) = 150 + 286 + 3000 = 3436 I/O