University of El Oued Department of Computer Science
2nd Y Master Academic Year: 2022/2023

Answers of Final Exam of Distributed Databases

Exercise 01 (7 pts)

A) What is the estimated cost in terms of read and write pages (I/O accesses) required in the following situations? Explain why?

1. Sort a table that is stored on N pages with a buffer of size B pages such as B<N.

   $2N(1+Log_{B-1}(N/B))$

2. Join two tables stored in N and M pages respectively using nested loop join algorithm with a buffer of size B pages such as B>M+2 and B>N+2.

   M+N

3. Join two tables where each one of them is stored on 1 page using sort-merge join algorithm.

B) What is the minimum amount of buffer size required to calculate R⋈S in a cost=[R]+[S] (size of R + size of S).

2

C) What would the occupancy of each leaf node of a B^+-tree be if index entries were inserted in sorted order? Explain why?

From the definition of the B^+-Tree index, when a new entry is inserted into a full leaf node, a new leaf node will be added, where the old leaf node maintains half of (50%) entries, and the other half will be inserted with the new entry into the new leaf node. As the inserted data are sorted, the empty entries in leaf nodes will never be filled.

Therefore, all leaf nodes occupy 50% of their entries, except the last that occupy 50%+1 or more

D) Given a relation R horizontally fragmented into R1, R2, …, Rn. Show reconstruction, completeness, and disjointness on this fragmentation.

Reconstruction: R = R1 ∪ R2 ∪ … ∪ Rn

Completeness: R = R1 ∪ R2 ∪ … ∪ Rn

Disjointness: R1 ∩ R2 ∩ … ∩ Rn = Ø,

Exercise 02 (5 pts)

A Banking database contains the following relations among others:

Accounts (AccountNr, Balance), DebitCards (CardNr, AccountNr, CustomerId),
Customers (CustomerId, FullName, Ssn, Address, City),
Transactions (Site, TransactDate, TransactType, Amount, AccountNr, CardNr),

The relation **Accounts** is range-horizontally fragmented according to the attribute AccountNr. The relation **DebitCards**, as well as the relation **Transactions**, are range-horizontally fragmented according to the attribute CardNr (Debitcard's number) in the same manner. The relation **Customers** is range-horizontally fragmented according to the attribute FullName.

1. How to execute the following SQL clauses in an optimal manner?

   a] select sum(Amount)from Transactions where Amount> 1000.

Assume that Transactions is fragmented to T1, T2, … Tk

b] select count(*) from Customers where FullName like 'S%'.

As Customers relation is fragmented according to FullName, so it is possible to directly choose the fragment Ci according to the condition "FullName like 'S%'".

c] select max(balance) from Accounts where AccountNr < 1234.

As Accounts relation is fragmented according to AccountNr, so it is possible to directly choose the fragment Ai according to the condition "AccountNr < 1234".

2. Give the optimized execution plan of the query:

```
select T.site, T.TransactDate, T.TransactType, T.amount, D.CustomerId
from DebitCards D, Transactions T, Customers C
 where C.FullName like 'S%' and C.CustomerId = D.CustomerId and D.CardNr = T.CardNr
```

Assume that: Transactions is fragmented to T1, T2, … Tm
DebitCards is fragmented to D1, D2, … Dm
Customers is fragmented to C1, C2, … Cn

the above query can be written in relational algebra as

$$\sigma_{\text{FullName like 'S%'}}\ \textbf{(Ci)} \bowtie (T1 \cup \textbf{T1} \cup \ldots \cup \textbf{Tm}) \bowtie (D1 \cup \textbf{D1} \cup \ldots \cup \textbf{Dm})$$

We can apply the distributivity of join over union and associativity of joins between Ci and fragments of Transaction to filter them by join operator. Furthermore, because of DebitCards and Transactions relations are fragmented with the same criteria, for all i≠j, Ti⋈Dj= Ø, so only joins Ti⋈Dj / i=j will be maintained

So the optimized plan becomes:

$$\left((\sigma_{\text{FullName like 'S%'}}\ \textbf{(Ci)} \bowtie \textbf{T1}) \bowtie \textbf{D1}\right) \cup \left((\sigma_{\text{FullName like 'S%'}}\ \textbf{(Ci)} \bowtie \textbf{T2}) \bowtie \textbf{D2}\right)$$

$$\cup \ldots \cup \left((\sigma_{\text{FullName like 'S%'}}\ \textbf{(Ci)} \bowtie \textbf{Tm}) \bowtie \textbf{Dm}\right)$$

Exercise 03 (4 pts)

Given the following relations EMPLOYEE(ENR, ENAME, JOB, SALARY)
PROJECT(PNR, PNAME, LOCATION) ASSIGNMENT(ENR, PNR, DURATION)

Each of the three relations is stored on a different node.

Furthermore, the following statistics are known: card(EMPLOYEE)=1.000, card(ASSIGNMENT)=1.500, card(PROJECT)=200. The query is initiated at node NEMPLOYEE and the result must be returned there. The job selection is satisfied by 10% of the employees (SF = 0, 1); 25% of the employees work in no specific project.

Estimate the communication cost for the following query:

```
SELECT * FROM EMPLOYEE E, PROJECT P, ASSIGNMENT A
WHERE E.ENR=A.ENR AND P.PNR=A.PNR AND JOB='SW-Developer'
```

The best cost (we consider the worst case)

(1)---100—>(2) join in (2)

(3)---200—>(2) join in (2)

(2)----200—-(1) the result is sent back to (1)

So the best cost is 500

Exercise 04 (4 pts)

Let r and s be relations with no indices, and assume that the relations are not sorted. Assume also that r is stored in N pages and s in M pages.

1. Assuming infinite memory, what is the lowest cost (in terms of I/O operations) to compute: r ⋈ s using

a. block nested loop join algorithm

[r]+[s] because we can load all r pages and then load each s page ones

b. sort-merge join algorithm

[r]+[s] because we can load all r and s pages, sort them in memory then join by merge

2. What is the minimum amount of memory required for each algorithm while maintaining the same cost of the question 1?

For the block nested loop join algorithm

the minimum amount of memory required is $\min([r],[s])+2$

because we load at first all pages of the relation having a lower number of pages, then we need a page of buffer to load the second relation page per page and a second page in the buffer to put join results.

For the sort-merge join algorithm,

the minimum amount of memory required is $[r]+[s]+1$

because we load all pages of both r and s relations to sort each of them in memory (so we need [r]+[s]). We need another page in the buffer to put join results.