Monsoon Mid-Semester Examination, Session 2023-24

[SOLUTION]

Examination & Semester: M.Tech (Computer Science & Engineering) I Semester
Subject: Advanced DBMS (CSC502) **Time:** 2 Hours
Instructions: **Max. Marks:** 56

1. (a)	Give an expression in the **Relational Algebra** (**<u>using Basic Operators only</u>**) for each request on schema: *employee (<u>person_name</u>, street, city)* *works (<u>person_name</u>, company_name, salary)* *company (<u>company_name</u>, city)* *manages (<u>person_name</u>, manager_name)* (i) Modify the database so that Amit now lives in Dhanbad city. (ii) Give all employees of Facebook a 10 percent salary raise. (iii) Find the names of all employees in this database who live in the same city as the company for which they work. (iv) Give all managers in this database a 10 percent salary raise. (v) Assume the companies may be located in several cities. Find all companies located in every city in which Amazon is located.	**5**

<u>Solution:</u>

Sample Answer:

(i) Modify the database so that Amit now lives in Dhanbad city.

$t_1 \leftarrow \sigma_{\text{person\_name = 'Amit'}}(\text{employee})$

$t_2 \leftarrow \Pi_{\text{person\_name, street, city = 'Dhanbad'}}(\sigma_{\text{person\_name = 'Amit'}}(\text{employee}))$

$\text{employee} \leftarrow (\text{employee} - t_1) \cup t_2$

(ii) Give all employees of Facebook a 10 percent salary raise.

$t_1 \leftarrow \sigma_{\text{company\_name = 'Facebook'}}(\text{works})$

$t_2 \leftarrow \Pi_{\text{person\_name, company\_name, salary = salary*1.10}}(\sigma_{\text{company\_name = 'Facebook'}}(\text{works}))$

$\text{works} \leftarrow (\text{works} - t_1) \cup t_2$

(iii) Find the names of all employees in this database who live in the same city as the company for which they work.

$\Pi_{\text{employee\_person\_name}}(\sigma_{\text{employee.person\_name = works.person\_name} \wedge \text{works.company\_name = company.company\_name} \wedge \text{employee.city = company.city}}(\text{employee} \times \text{works} \times \text{company}))$

(iv) Give all managers in this database a 10 percent salary raise.

$t_1 \leftarrow \Pi_{\text{works.person\_name, company\_name, salary}}(\sigma_{\text{works.person\_name = manages.person\_name}}(\text{works} \times \text{manages}))$

$t_2 \leftarrow \Pi_{\text{works.person\_name, company\_name, salary = salary*1.10}}(\sigma_{\text{works.person\_name = manages.person\_name}}(\text{works} \times \text{manages}))$

$\text{works} \leftarrow (\text{works} - t_1) \cup t_2$

(v) Assume the companies may be located in several cities. Find all companies located in every city in which Amazon is located.

$t_1 \leftarrow \Pi_{\text{city}}(\sigma_{\text{company\_name = 'Amazon'}}(\text{company}))$

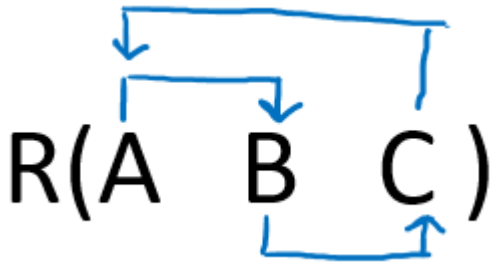
$t_2 \leftarrow \Pi_{\text{company\_name}}(\text{company})$

$\text{OUTPUT} \leftarrow t_2 - \Pi_{\text{company\_name}}((t_2 \times t_1) - \text{company})$

1. (b)	Given a relation R(A, B, C) and Functional Dependency set FD = { A → B, B → C, and C → A}, determine given R is in which normal form?	**3**

Solution:

Let us construct an arrow diagram on R using FD to calculate the candidate key.



From the above arrow diagram on R, we can see that all the attributes are determined by all the attributes of the given FD, hence we will check all the attributes (i.e., A, B, and C) for candidate keys

Let us calculate the closure of A

A^+ = ABC (from the closure method we studied earlier)

Since closure A contains all the attributes of R, hence A is the Candidate key.

Let us calculate the closure of B

B^+ = BAC (from the closure method we studied earlier)

Since closure B contains all the attributes of R, hence B is the Candidate key.

Let us calculate the closure of C

C^+ = CAB (from the closure method we studied earlier)

Since closure C contains all the attributes of R, hence C is the Candidate key.

Hence three Candidate keys are: **A B and C**

Since R has 3 attributes: - A, B and C, Candidate Keys are A, B and C. Therefore, prime attributes (part of candidate key) are A, B, C while there is no non-prime attribute

Given FD are { A → B, B → C, and C → A } and Super Key / Candidate Key is A, B and C

a. FD: **A** → **B** satisfy the definition of BCNF, as A is Super Key, we check other FD for BCNF

b. FD: **B** → **C** satisfy the definition of BCNF, as B is Super Key, we check other FD for BCNF

c. FD: **C à A** satisfy the definition of BCNF, as C is Super Key

Since there were only three FD's and all FD: { A → B, B → C and C → A } satisfy BCNF, hence the highest normal form is BCNF.

Therefore R(A, B, C) is in BCNF.

2. (a)	Show the steps to remove extraneous attributes from the functional dependency. Find the canonical cover of the given F. F = {A → BC, B → CE, A → E, AC → H, D → B}	**5**

Solution:

Steps to remove Extraneous Attributes from the functional dependency:

1. If A∈β, to check if 'A' is extraneous consider the set **(1)**
   F` = (F – { α → β }) U { α → (β – A)}
   And check if α → A can be inferred from F`.
   To do so, compute α^+ under F`.
   If α^+ includes A, then A is extraneous in β.

2. If A∈α, to check if 'A' is extraneous, let γ = α – {A}, and check if γ → β can be inferred from F. **(1)**
   To do so, compute γ^+ under F.
   If γ^+ includes all attributes in β, then A is extraneous in α.

Canonical Cover of F **(3)**

F = {A→BC, B→CE, A→E, AC→H, D→B}

Fc = {A→BC, B→CE, A→E, AC→H, D→B}

Fc = {A→BCE, B→CE, AC→H, D→B} ……. (Union Rule in A→BC & A→E)

Check 'C' is extraneous in AC→H ……. YES (Remove C from AC→H)

Fc = {A→BCE, B→CE, A→H, D→B}

Fc = {A→BCEH, B→CE, D→B} ……. (Union Rule in A→BCE & A→H)

Check 'C' is extraneous in B→CE …….. NO

Fc = {A→BCEH, B→CE, D→B}

Check 'E' is extraneous in B→CE …….. NO

Fc = {A→BCEH, B→CE, D→B}

Check 'B' is extraneous in A→BCEH …….. NO

Fc = {A→BCEH, B→CE, D→B}

Check ‘C’ is extraneous in A→BCEH …….. YES (Remove C from A→BCEH)

Fc = {A→BEH, B→CE, D→B}

Check ‘E’ is extraneous in A→BCEH …….. YES (Remove E from A→BCEH)

Fc = {A→BH, B→CE, D→B}

Check ‘H’ is extraneous in A→BH …….. NO

Fc = {A→BH, B→CE, D→B}

2. (b)	Suppose we have a relation R(a, b, c, d, e) and there are at least 1000 distinct values for each of the attributes. Consider each of the following query workloads, independently of each other. If it is possible to speed it up significantly by adding up to two additional indexes to relation R, specify for each index (1) which attribute or set of attributes form the search key of the index, (2) if the index should be clustered or unclustered, (3) if the index should be a hash-based index or a B+-tree. You may add at most two new indexes. If adding a new index would not make a significant difference, you should say so. Give a brief justification for your answers. (i) 100,000 queries have the form: select * from R where b<? 10,000 queries have the form: select * from R where c=? (ii) 100,000 queries have the form: select * from R where b<? and c=? 10,000 queries have the form: select * from R where d=? 1,000 queries have the form: select * from R where a=?	**3**

Solution:

(i) For the first query, since we need efficient range-queries on R(b), we definitely want a **clustered**, **B+-tree index on R(b)**.

For the second query, **an index on R(c)** will help. It can be either a **B+-tree or a hash-based index** since queries look-up specific key values. The index must be **un-clustered** since the index on R(b) is clustered.

(1.5)

(ii) For the first query, a **clustered B+-tree index on R(c,b)** would be most helpful since we could use it to look-up all data items that match both the given value on ‘c’ and the range on ‘b’.

Since we can only add a second index, we will favor the most frequent query and add **an index on R(d)**. As in the question above, this index must be **un-clustered** and can be **either a B+-tree or a hash-based index**.

(1.5)

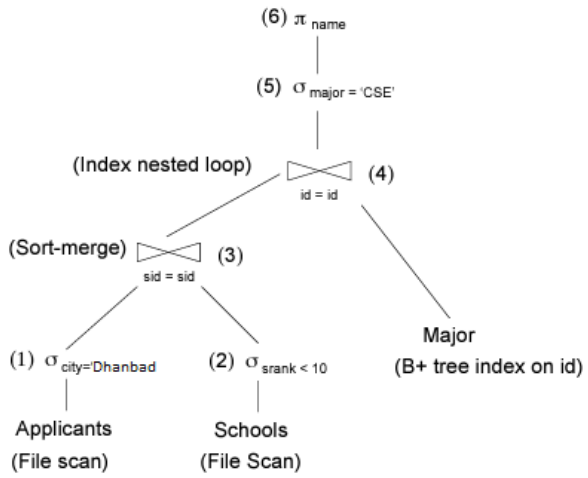
3. (a)	Consider the following SQL query that finds all applicant who want to major in CSE, live in Dhanbad, and go to a school ranked better than10 (i.e., rank<10). SELECT A.name FROM Applicants A, Schools S, Major M	**5**

WHERE A.sid=S.sid AND A.id=M.id AND A.city='Dhanbad' AND S.rank<10 AND M.major='CSE'

Details of relations used are:
Applicants(id, name, city, sid) : Cardinality: 2,000 and Number of pages: 100
Schools(sid, sname, srank) : Cardinality: 100 and Number of pages: 10
Major(id, major) : Cardinality: 3,000 and Number of pages: 200

And assuming:
• Each school has a unique rank number (srank value) between 1 and 100.
• There are 20 different cities.
• Applicants.sid is a foreign key that references Schools.sid.
• Major.id is a foreign key that references Applicants.id.
• There is an unclustered, secondary B+ tree index on Major.id and all index pages are in memory.

What is the cost of the query plan below? Count only the number of page I/Os.



Solution:

The total cost of this query plan is 119 I/Os.

Is computed as follows:

(1) **The cost of scanning Applicants is 100 I/Os.** The output of the selection operator is 100/20 = 5 pages or 2000/20 = 100 tuples. [Assume uniform distribution]. **(1)**

(2) The cost of scanning Schools is 10 I/Os. The selectivity of the predicate on rank is (10−1)/99 = 0.09. The output is thus $0.09*10 \approx 1$ page or $0.09*100 \approx 9$ tuples. **(1)**

(3) Given that the input to this operator is only six pages, we can do an in-memory sort-merge join. The cardinality of the output will be 9 tuples. There are two ways to compute this: **(1)**
(a) $(100*9)/(\max(100,9))=9$ or
(b) consider that this is a key-foreign key join and each applicant can match with at most one school but keep in mind that the predicates on city and rank were independent, hence only 0.9 of the applicants end-up with a matching school.

(4) The index-nested loop join must perform one look-up for each input tuple in the outer relation. We assume that each student only declares a handful of majors, so all the matches fit in one page. **The cost of this operator is thus 9 I/Os.** **(2)**
(5) and (6) are done on-the-fly, so there are no I/Os associated with these operators.

3. (b)	Consider an institute and the relation *Registered(Stu_Admn_no, Course_No, Session, Year, Units, Grade)* contains the grades for the courses completed by students during the last 20 years. For simplicity, assume that there are 25,000 students enrolled each session, and that each student takes four courses per session, and that there are four sessions each year. Then we get a total of 8,000,000 records. If 10,000 new students enter institute every year, we can assume that in *registered* there are 200,000 different students, each identified by its Stu_Admn_no. On the average, a student took 40 different courses. The file blocks hold 2048 bytes and each took tuple requires 50 bytes. The table has a primary index on Stu_Admn_no. If the Stu_Admn_no index is a B+ tree with order n = 101, how many levels does the B+ tree use, in the worst case. Justify your answer.	**3**

Solution:

There are 200,000 different student on an average in 20 years. Each student has unique Stu_Admn_no.
The table has primary index on Stu_Admn_no which means it has 200000 tuples.
So, total number of levels $\lceil \log_{101} 200000 \rceil = 3$ **(3)**

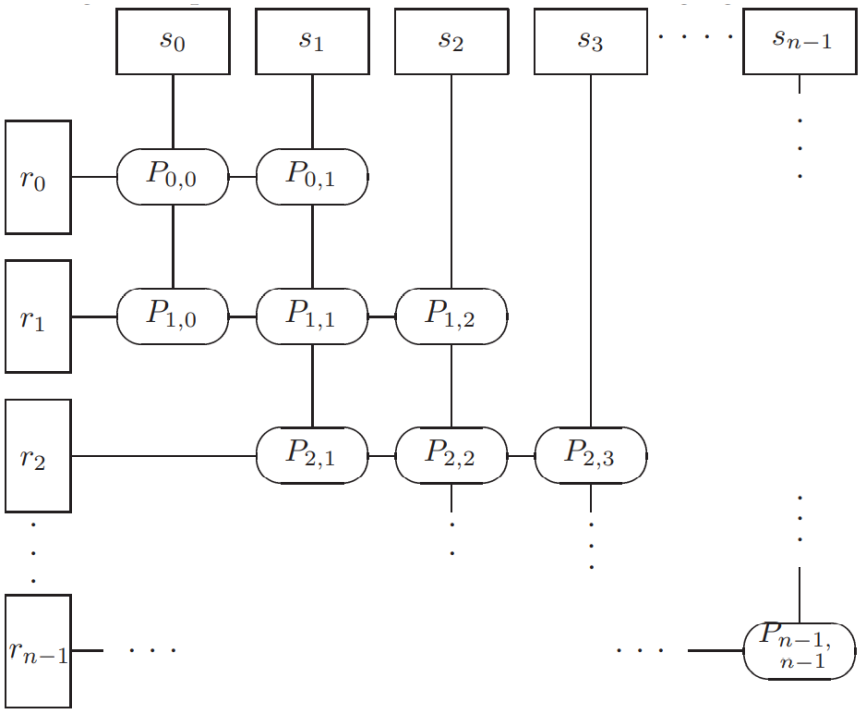
OR

0th level has 1 node (root).
1st level has 101∗101 node.
2nd level has 101∗101 node.
3rd level has 101∗101∗101 node.
So, 1+101+101∗101+101∗101∗101 this will include all the rows which need at least 3 levels considering root at 0th level. **(3)**

4. (a)	Consider join processing using symmetric fragment and replicate with range partitioning. How can you optimize the evaluation if the join condition is of the form $\mid r.A - s.B \mid \leq k$, where k is a small constant. Here, $\mid x \mid$ denotes the absolute value of x. A join with such a join condition is called a band join.	**5**

Solution:

Relation r is partitioned into n partitions, r_0, r_1,...,r_{n-1}, and s is also partitioned into n partitions, s_0, s_1,...,s_{n-1}.
Range Partitioning with range vector as [k, 2k, 3k, ….., nk] will be chosen. **(2)**
The partitions are replicated and assigned to processors as shown in the following figure. **(2)**

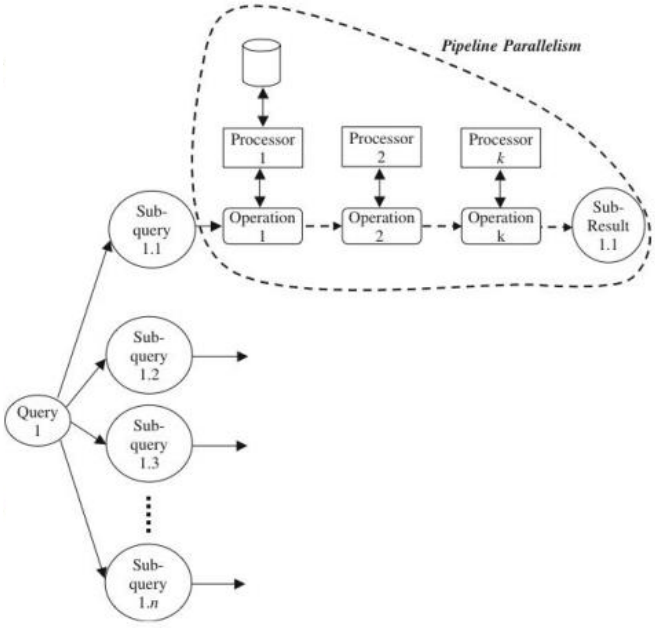


Each fragment is replicated on 3 processors only, unlike in the general case where it is replicated on n processors.
The number of processors required is now approximately 3n, instead of n^2 in the general case.
Therefore given the same number of processors, we can partition the relations into more fragments with this optimization, thus making each local join faster. **(1)**

4. (b)	Differentiate been pipelined and independent parallelism. Set up a pipeline architecture to compute three joins in parallel.	**3**

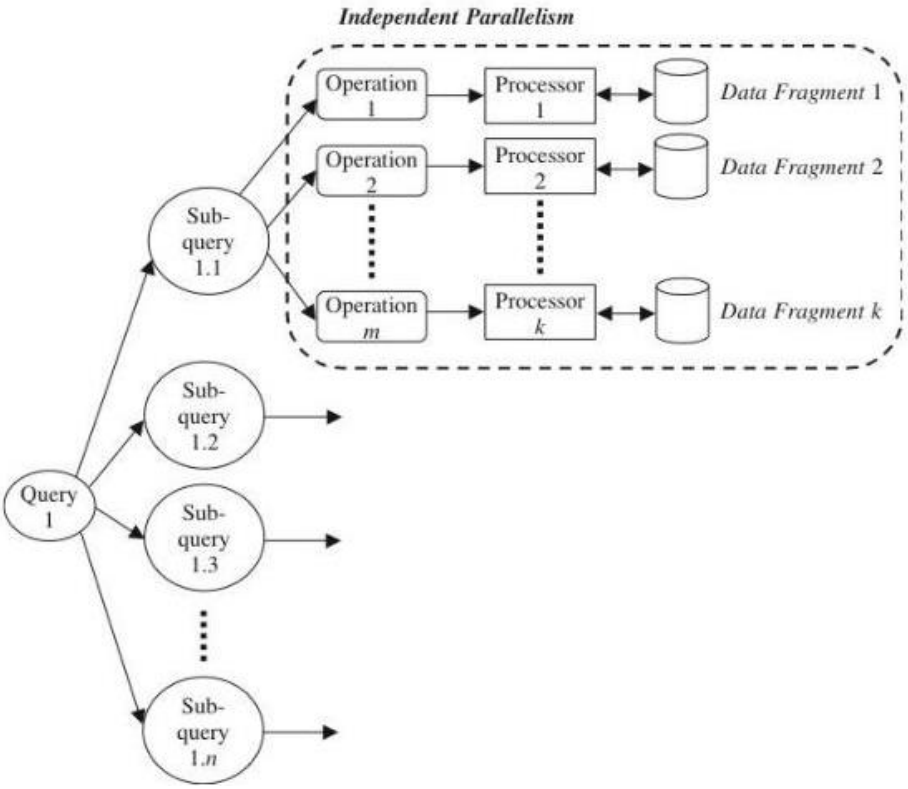
Solution:

Pipelined Parallelism: **(1)**



- Output record of one operation A are consumed by a second operation B, even before the first operation has produced the entire set of records in its output.
- Multiple operations form some sort of assembly line to manufacture the query results.
- Useful with a small number of processors, but does not scale up well.

Independent Parallelism: **(1)**



- Operations in a query that do not depend on another are executed in parallel.
- Does not provide high degree of parallelism.

Consider a join of four relations **(1)**

R1 ⋈ R2 ⋈ R3 ⋈ R4

Set up a pipeline that computes the three joins in parallel:

- ➔ Let P1 be assigned the computation of temp1 = R1 ⋈ R2
- ➔ And P2 be assigned the computation of temp2 = temp1 ⋈ R3
- ➔ And P3 be assigned the computation of temp2 ⋈ R4

Each of these operations can execute in parallel, sending result tuples it computes to the next operation even as it is computing further results.