Monsoon Semester Examination Solution, Session 2023-24

Examination & Semester: **M.Tech (Computer Science & Engineering) I Semester**
Subject: **Advanced DBMS (CSC502)** Time: **3 Hours**
Instructions: Max. Marks: **100**

(a) Answer the questions serially and to the point.
(b) Assume a suitable value against any missing data.

Q. No.	Question	Marks
1. (a)	Consider the following relations: *Customer (CN, CS, CC); Account (AN, BN, BAL); Loan (LN, BN, AMT); Depositor (CN, AN); Borrower (CN, LN)* Write a query using relation algebra to "Find the names of all customers who have a loan at the bank." Rewrite the query to include not only the name, but also the city of residence for each customer. It has been observe that now one of the customer, say "Amit", no longer appears in the result, even though Amit does in fact have a loan from the bank. (i) Explain why Amit does not appear in the result. (ii) Suppose that you want Amit to appear in the result. How would you modify the database to achieve this effect? (iii) Again, suppose that you want Amit to appear in the result. Rewrite a query so that it accomplishes this desire without having to modify the database.	**5**

Solution:

Sample Query:

$\Pi_{\text{customer-name}}$ (borrower ⋈ loan ⋈ customer) **[1]**

OR

$\Pi_{\text{customer-name}}$ (borrower ⋈ customer)

OR

$\Pi_{\text{customer-name}}$ (borrower)

$\Pi_{\text{customer-name, customer-city}}$ (borrower ⋈ loan ⋈ customer) **[1]**

OR

$\Pi_{\text{customer-name, customer-city}}$ (borrower ⋈ customer)

(i) Although Amit does have a loan, no address is given for Amit in the customer relation. Since no tuple in customer joins with the Amit tuple of borrower, Amit does not appear in the result. **[1]**

(ii) The best solution is to insert Amit's address into the customer relation. If the address is unknown, null values may be used. If the database system does not support nulls, a special value may be used (such as unknown) for Amit's street and city. The special value chosen must not be a plausible name for an actual city or street. **[1]**

(iii) Sample Query:

$\Pi_{\text{customer-name, customer-city}}$ ((borrower ⋈ loan) ⟕ customer) **[1]**

OR

$\Pi_{\text{customer-name, customer-city}}$ (borrower ⟕ customer)

Q. No.	Question	Marks
1. (b)	Use Armstrong's axioms to prove the soundness of the decomposition rule. Explain how functional dependencies can be used to indicate the following [relation schema is provided in 1(a)]: (i) A one-to-one relationship set exists between entity sets Account and Customer.	**5**

	(ii) A many-to-one relationship set exists between entity sets Account and Customer.	

Solution:

The decomposition rule, and its derivation from Armstrong's axioms are given below: **[3]**
if $\alpha \rightarrow \beta\gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$.
$\alpha \rightarrow \beta\gamma$ given
$\beta\gamma \rightarrow \beta$ reflexivity rule
$\alpha \rightarrow \beta$ transitivity rule
$\beta\gamma \rightarrow \gamma$ reflexive rule
$\alpha \rightarrow \gamma$ transitive rule

Let Pk(r) denote the primary key attribute of relation r.

(i) The functional dependencies Pk(account)→Pk (customer) and Pk(customer) → Pk(account) indicate a one-to-one relationship because any two tuples with the same value for account must have the same value for customer, and any two tuples agreeing on customer must have the same value for account. **[1]**

(ii) The functional dependency Pk(account) → Pk(customer) indicates a many-to-one relationship since any account value which is repeated will have the same customer value, but many account values may have the same customer value. **[1]**

Q. No.	Question	Marks
1. (c)	Suppose you are given a relation R with four attributes ABCD. For each of the following sets of FDs, assuming those are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) Identify the best normal form that R satisfies (1NF, 2NF, 3NF, or BCNF). (c) If R is not in BCNF, decompose it into a set of BCNF relations that preserve the dependencies. Show each decomposition step (if any) clearly. (i) $C \rightarrow D, C \rightarrow A, B \rightarrow C$ (ii) $ABC \rightarrow D, D \rightarrow A$	**5**

Solution:

(i) Candidate keys: B **[0.5]**
R is in 2NF but not 3NF. **[1]**
C → D and C → A both cause violations of BCNF. One way to obtain a (lossless) join preserving decomposition is to decompose R into AC, BC, and CD. **[1]**

(ii) Candidate keys: ABC, BCD **[1]**
R is in 3NF but not BCNF. **[1]**
ABCD is not in BCNF since D → A and D is not a key. However if we split up R as AD, BCD we cannot preserve the dependency ABC → D. So there is no BCNF decomposition. **[0.5]**

Q. No.	Question	Marks
1. (d)	Answer the following, with proper justification (Assume the height numbering starts from 0.): (i) What is the maximum fan-out of a B+ tree of an order x+1? (ii) What is the maximum number of keys an order 2 B+ tree with height 2 can store? (iii) For a height 3 B+ tree (leaf pages hold pointers to the corresponding records), how many I/Os are required for an equality search on the index key given that the index key is a primary key?	**5**

Solution:

(i) Practically it is same as order of B+ tree i.e., x+1. **[1]**

(ii) 4. An order of 2 means the fan-out is 2. At height 0, there is one node. At height 1, there are 1*2 = 2 nodes. At height 2, there are 2*2 = 4 nodes. Each node can store 1 keys. So 4*1 = 4 keys. **[2]**

(iii) 5 I/Os. We need 4 I/Os to traverse to a leaf page and 1 I/O to go to the actual page in the file. **[2]**

Q. No.	Question	Marks
2. (a)	Due to hot weather, our machine has forgotten how to sort. Luckily, we have indexes on the join columns between R and S. R's index is an un-clustered index with a height of 2 and contains 10 leaf pages. S's index is a clustered index with a height of 2 and contains 12 leaf pages. R consists of 30 pages with 4 records on each page and S consists of 40 pages with 5 records on each page. What is the average cost of performing Sort Merge Join on R and S? You can assume that resetting to a record will never require accessing a previous leaf page or a previous data page and that there are enough memory pages to satisfy any operation.	**5**

Solution:

Sort Merge Join on R, S normally iterates through both relations, compares records, and costs N+M I/Os where N = # of pages of R and M = # of pages of S. This same logic can be applied for indexes since leaf pages can be iterated over as each page contains a pointer to the next leaf page.

Iterating over R's sorted records takes a total of 132 I/Os: 3 for traversing to a leaf, 9 for traversing all the leaf pages, and 30*4 for accessing every record since the index is un-clustered. **[2]**

Iterating over S's sorted records takes a total of 54 I/Os: 3 for traversing to a leaf, 11 for traversing the data pages, and 40 for accessing every data page. **[2]**

The total I/O cost for the average case = 186 I/Os. **[1]**

Q. No.	Question	Marks
2. (b)	Consider a relation 's' over the attributes A and B with the following characteristics: • 7,000 tuples with 70 tuples per page • A hash index on attribute A • The values that the attribute A takes in relation s are integers that are uniformly distributed in the range 1–200. (i) Assuming that the aforesaid index on A is unclustered and it requires 2 I/Os to locate right bucket, estimate the number of disk accesses needed to compute the query σA=18(s). (ii) What would be the cost estimate if the index were clustered and it requires 4 I/Os to locate right bucket?	**5**

Solution:

The number of tuples selected from s is expected to be 7000/200 = 35.

(i) If the index is unclustered, every tuple can potentially lead to a separate I/O. So, the cost will be ceiling(2 + 35) = 37. **[3]**

(ii) If the index is clustered, then 35 tuples can span at most 2 pages (2 rather than 1 because they might cluster around a page boundary), so the I/O cost would be ceiling(4+2)=4. **[2]**

Q. No.	Question	Marks
2. (c)	How does the quorum consensus protocol works? Can we say that the quorum consensus protocol is a generalization of biased and majority based protocol? Explain the same with a suitable example.	**5**

Solution:

Each site is assigned a weight. Let S be the total of all site weights. **[0.5]**
Choose two values read quorum Qr and write quorum Qw

- Such that Qr + Qw > S and 2*Qw > S **[1]**
- Quorums can be chosen (and S computed) separately for each item. **[0.5]**

Each read must lock enough replicas that the sum of the site with Qr. **[0.5]**
Each write must lock enough replicas that the sum of the site with Qw. **[0.5]**

Let Qr = 1, Qw = 4, site weight = 1 and total weight of sites (S) = 4 **[2]**

Read Lock: Requires lock on one site (say S1)
Write Lock: Requires lock on four sites (say S1 to S4)
This is the implementation of Biased protocol. At the same time, if we make read and write quorum with the same value 3, then it resembles the implementation of Majority based protocol. This is the reason why quorum consensus protocol is mentioned as the generalization of both majority and biased protocols.

Q. No.	Question	Marks
3. (a)	Briefly explain the concepts of Two-level serializability in the multi-database systems. Also emphasize on the restrictions that can be made for the better transaction management in the multi-database systems.	**5**

Solution:

Two-level serializability: **[2]**

1. Each DBMS ensures local serializability among its local transactions, including those that are part of a global transaction.
2. The multidatabase ensures serializability among global transactions alone – ignoring the orderings induced by local transactions.

The restrictions that can be made for the better transaction management in the multi-database systems:

(i) Global-read protocol: It ensures strong corrections, if all these conditions hold: **[1]**
   a. Local Transaction access only local data items.
   b. Global transaction access global data items and may read local data items.
   c. No consistency constraint's between local and global data items.

(ii) Local-read protocol: It ensures strong corrections, if all these conditions hold: **[1]**
   a. Local Transaction access to local data items and may read global data items stored at the site.
   b. Global transaction access only global data items.
   c. No transaction may have a value dependency.

(iii) Global-read-write/Local-read protocol: It ensures strong corrections, if all these conditions hold: **[1]**
   a. Local Transaction access to local data items and may read global data items stored at the site.
   b. Global transaction access to both local and global data items.
   c. No consistency constraint's between local and global data items.
   d. No transaction may have a value dependency.

Q. No.	Question	Marks
3. (b)	Explain, with a suitable example, the concepts of type and table inheritance of object-based databases.	**5**

Solution:

Type Inheritance: **[2.5]**

- Syntax for creating inheritance (use of under clause)
- Single and multiple inheritance example.
- Use of final/not final, for creating subtype.

Table Inheritance: **[2.5]**

- Syntax for creating table inheritance (uses of of clause and under clause).
- Tuples added to a sub-table are automatically visible to queries on the super table. Use of only in a query to override this behaviour.
- Each tuple of the super table can correspond to at most one tuple in each of the sub-tables.

Q. No.	Question	Marks
3. (c)	NoSQL do not offer support for ACID properties of transactions. How the databases like MongoDB and Cassandra manage the concurrent transactions without the ACID properties? Justify your answer.	**5**

Solution:

Follow CAP Theorem instead of ACID property. **[1]**
A distributed system can deliver only two of three desired characteristics: Consistency (C), Availability (A) and Partition Tolerance (P).

MongoDB is CP database – This database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved. **[2]**

Cassandra is AP database – This database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.). **[2]**

Q. No.	Question	Marks
4. (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}	**6**

Solution:

Steps to remove Extraneous Attributes from the functional dependency:

1. If A$\in$β, to check if 'A' is extraneous consider the set **[1.5]**
   $F` = (F – \{ \alpha \rightarrow \beta \}) \cup \{ \alpha \rightarrow (\beta – A)\}$
   And check if $\alpha \rightarrow A$ 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$\in$α, to check if 'A' is extraneous, let $\gamma = \alpha – \{A\}$, and check if $\gamma \rightarrow \beta$ can be inferred from F. **[1.5]**

   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}

Q. No.	Question	Marks
4. (b)	The keys 1, 3, 12, 4, 25, 6, 18, 20, and 8 are inserted in to an empty hash table of length 10 using open addressing with hash function as $H(i) = i^2 \bmod 10$ and linear probing. What is the resultant hash table and find the maximum probe value?	**5**

Solution:

All other elements will go to the respective place without any collision.
Except
For 6 (desired location is 6) but it will go to next vacant space i.e., at 7.
For 18 (desired location is 4) but it will go to next vacant space i.e., at 8.
For 8 (desired location is 4) but it will go to next vacant space i.e., at 2.

Index Number	Value
0	**20**
1	**1**
2	**8**
3	
4	**12**
5	**25**
6	**4**
7	**6**
8	**18**
9	**3**

[3]

So,
The probe value for inserting 6 is 2.
The probe value for inserting 18 is 5.
The probe value for inserting 8 is 9.

Therefore, maximum probe value is 9. **[2]**

Q. No.	Question	Marks
4. (c)	Consider these relations with the following properties: r(A, B, C) - 30,000 tuples and 25 tuples fit on 1 block s(C, D, E) - 60,000 tuples and 30 tuples fit on 1 block (i) Estimate the number of disk block accesses required for a natural join of r and s using a nested-loop join if r is used as the outer relation. (ii) Estimate the number of disk block accesses required for a natural join of r and s using a block nested-loop join if s is used as the outer relation. Assume that there are more than 2000 memory buffers available to facilitate this operation, where each memory buffer can buffer one disk block.	**4**

Solution:

(i) Total no of block transfer required $= n_r * b_s + b_r$
$= 30000*2000 + 1200$
$= 60001200$ **[2]**

(ii) Total no of block transfer required $= b_s * b_r + b_s$
$= 2000*1200 + 2000$
$= 2402000$ **[2]**

Q. No.	Question	Marks
5. (a)	Briefly explain the different forms of parallelism. What form of parallelism is likely to be the most important for each of the following tasks [also justify your claim]: (i) Increasing the throughput of a system with many small queries. (ii) Increasing the throughput of a system with a few large queries, when the number of disks and processors is large.	**6**

Solution:

Different forms of parallelism: Intra-query parallelism, Inter-query parallelism, Intra-operation parallelism, and Intra-operation parallelism. **[2]**

i) When there are many small queries, inter-query parallelism gives good throughput. Parallelizing each of these small queries would increase the initiation overhead, without any significant reduction in response time. **[2]**

ii) With a few large queries, intra-query parallelism is essential to get fast response times. Given that there are large number of processors and disks, only intra-operation parallelism can take advantage of the parallel hardware – for queries typically have few operations, but each one needs to process a large number of tuples. **[2]**

Q. No.	Question	Marks
5. (b)	Design an efficient query plan for the following distributed query: An application at site B wants to compute a join (Student $\bowtie_{Id=StudId}$ Gradesheet), where Student(Id, Major) is at site B and Gradesheet(StudId, CrsCode) is at site C. The result should be returned to B. Assume that semijoin is not used. Also assume that • The various lengths are: – Id and StudId are 8 bytes long; – Major is 3 bytes long; – CrsCode is 6 bytes long. • Student has 15,000 tuples. • 6,000 students are registered for at least one course. On the average, each student is registered for 5 courses.	**5**

Solution:

Gradesheet has 6,000*5=30,000 tuples, each 14 (8+6) bytes long.
Each Student tuple has 11 (8+3) bytes.
The join has 8+6+3=17 bytes in each tuple.

1. If we send Student to site C, compute the join there, and then send the result to site B, we have to send 675,000 bytes ($= 15,000 * 11 + 30,000 * 17$). **[2]**

2. If we send Gradesheet to site B and compute the join there, we have to send 420,000 bytes ($= 30,000 * 14$). **[2]**

Thus, alternative 2 is better. **[1]**

Q. No.	Question	Marks
5. (c)	Consider the two-phase commit (2PC) protocol with write-ahead undo + redo logging [only the log file needs to be flushed to disk at the time of transaction commit]. Assume we have a system where the only failures involve hosts halting with their disks and logs intact, followed (potentially) by reboots, with no network message loss. (i) Suppose there is a coordinator C and two participants P1 and P2. Suppose we have the following sequence of events in a 2PC: *C sends Prepare Transaction T1 to P1, P2* *P1 sends Prepared to C* *P2 sends No to C* Write an appropriate next message following 2PC, ignoring potential redundant retransmissions. (ii) Suppose we have the following sequence of events in a 2PC: *C sends Prepare Transaction T2 to P1,P2* *P1 sends Prepared to C* *P2 sends Prepared to C* *P2 crashes* *P2 comes back online* *C sends Commit Transaction T2 to P1, P2* *P1 commits, sends DONE to C* Assuming standard 2PC operation in the face of the above scenario, what we can expect P2 to do next?	**4**

Solution:

(i) C sends Abort T1 to P1, P2 **[2]**
(i) P2 commits (T2), sends DONE to C **[2]**

Q. No.	Question	Marks
6. (a)	Explain the following with respect to Cassandra database: (i) Define and create a keyspace with replication. (ii) Explain the concepts of partition and clustering key. Demonstrate the same with a suitable example.	**6**

Solution:

(i) Keyspace is the outermost container for data in Cassandra. **[1]**
The basic attributes of a keyspace are:
Replication Factor – It is the number of machines in the cluster that will recei9ve copies of the same data.
Replica-placement Strategy – It is nothing but the strategy to place replicas in the ring. Strategies: Simple Strategy (rack-aware strategy: replication in one Data Centre only) and Network Topology Strategy (Datacenter-shared strategy: replication on multiple data center).

```
CREATE KEYSPACE Keyspace_Name WITH replication =
{ 'class': 'SimpleStrategy/NetworkTopologyStrategy',
  'replication_factor': count};
```
[2]

(ii) Primary Key = Partition Key + Clustering Key

Partition Key: How data is distributed across nodes? **[0.5]**
Clustering Key: How data is stored on a single node? **[0.5]**

Explanation of both is expected. **[1]**

Steps to create such keys: **[1]**

```
CREATE TABLE Name_of_Table
{
    Attribute1      type1,
    Attribute2      type1,
```

```
    Attribute3      type2,
    Attribute4      type3,
    Attribute5      type2,
    Primary Key ((Attribute1, Attribute2), Attribute3, Attribute4)
}
WITH CLUSTERING ORDER BY (Attribute3 ASC, Attribute4 DESC);
```

Partition Key: Attribute1, Attribute2
Clustering Key: Attribute3, Attribute4

Q. No.	Question	Marks
6. (b)	What is the need of a persistent programming language? Explain the different ways of making an object persistent.	**5**

Solution:

Need of a Persistent programming language: **[1]**

- The query language is fully integrated with the host language, and both share the same type system.
- The programmer can manipulate persistent data without writing code explicitly to fetch it in to memory or share it back to database.

Object can be made persistent by [***Explanation is expected for all four.***]: **[4]**

- Persistence by class
- Persistence by creation
- Persistence by marking
- Persistence by reachability

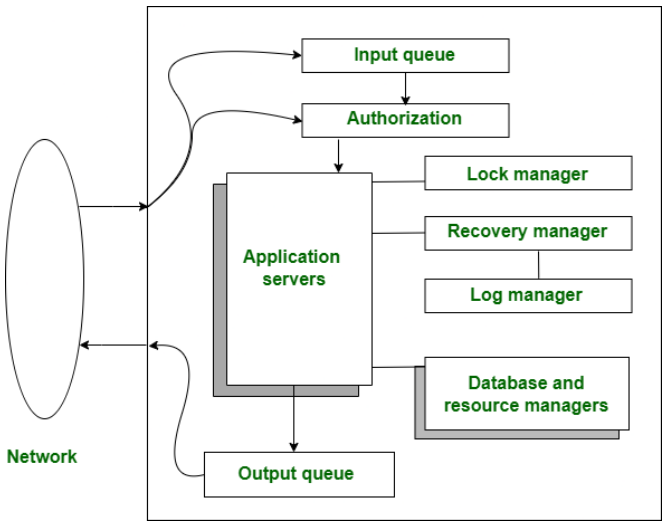
Q. No.	Question	Marks
6. (c)	What is a need of TP monitor? Draw and explain the detailed structure of TP monitor.	**4**

Solution:

Need of TP monitor: **[1]**

- To support large numbers of terminals from a single process.
- It provides infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers.

Structure of TP monitor: **[1]**



Explanation of above structure is expected. **[2]**

Q. No.	Question	Marks
7. (a)	Consider the relations r_1(A, B, C), r_2(C, D, E), and r_3(E, F), with primary keys A, C, and E respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of r1⋈r2⋈r3, and give an efficient strategy for computing the join.	**4**

Solution:

The relation resulting from the join of r1, r2, and r3 will be the same no matter which way we join them, due to the associative and commutative properties of joins. **[2]**

So we will consider the size based on the strategy of ((r1 ⋈ r2) ⋈ r3). Joining r1 with r2 will yield a relation of at most 1000 tuples, since C is a key for r2. Likewise, joining that result with r3 will yield a relation of at most 1000 tuples because E is a key for r3. Therefore the final relation will have at most 1000 tuples.

OR

So we will consider the size based on the strategy of (r1 ⋈ (r2 ⋈ r3)). Joining r2 with r2 will yield a relation of at most 750 tuples, since E is a key for r3. Likewise, joining that result with r1 will yield a relation of at most 750 tuples because C is a key for r1. Therefore the final relation will have at most 750 tuples.

An efficient strategy for computing this join would be to create an index on attribute C for relation r2 and on E for r3. Then for each tuple in r1, we do the following: **[2]**

a. Use the index for r2 to look up at most one tuple which matches the C value of r1.

b. Use the created index on E to look up in r3 at most one tuple which matches the unique value for E in r2

Q. No.	Question	Marks
7. (b)	Write down the sequence of steps (along with the name of the equivalence rules used) needed to transform $\pi_A((R \bowtie_{B=C} S) \bowtie_{D=E} T)$ into $\pi_A((\pi_E(T) \bowtie_{E=D} \pi_{ACD}(S)) \bowtie_{C=B} R)$. List the attributes that each of the schemas R, S, and T must have and the attributes that each (or some) of these schemas must not have in order for the above transformation to be correct.	**6**

Solution:

Sequence of steps: **[3]**

(a) Associativity of the join: $\pi_A(R \bowtie_{B=C} (S \bowtie_{D=E} T))$
(b) Commutativity of the join: $\pi_A((S \bowtie_{D=E} T) \bowtie_{C=B} R)$
(c) Commutativity of the join: $\pi_A((T \bowtie_{E=D} S) \bowtie_{C=B} R)$
(d) Pushing projection π_A to the first operand of the join: $\pi_A(\pi_{AC}((T \bowtie_{E=D} S)) \bowtie_{C=B} R)$
(e) Pushing projection to the first operand in the innermost join: $\pi_A(\pi_{AC}((\pi_E(T) \bowtie_{E=D} S)) \bowtie_{C=B} R)$. This is possible if AC are the attributes of S and not of T.
(f) Pushing projection to the second operand in the innermost join: $\pi_A(\pi_{AC}((\pi_E(T) \bowtie_{E=D} \pi_{ACD}(S))) \bowtie_{C=B} R)$.
(g) Reverse of pushing a projection: $\pi_A((\pi_E(T) \bowtie_{E=D} \pi_{ACD}(S)) \bowtie_{C=B} R)$

- R must have: B (because of the join) **[0.5]**
- S must have: ACD (because of π_{ACD}) **[0.5]**
- T must have: E (because of π_E) **[0.5]**
- These schemas should not have identically named attributes, because otherwise it will not be clear which of the two identically named attributes will be renamed in the joins. In particular, T should not have A and C, because $\pi_{ACD}(S)$ clearly suggests that it is expected that S will have the attribute A that will survive for the outermost projection π_A to make sense, and the attribute C, which should survive in order for the join with R to make sense. **[1.5]**

Q. No.	Question	Marks
8. (a)	Consider a parallel DBMS in which each relation is stored by horizontally partitioning its tuples across all disks. *Employees(eid: integer, did: integer, sal: real)* *Departments(did: integer, mgrid: integer, budget: integer)* The *mgrid* field of Departments is the *eid* of the manager. Each relation contains 20-byte tuples, and the *sal* and *budget* fields both contain uniformly distributed values in the range 0 to 1,000,000. The Employees relation contains 100,000 pages, the Departments relation contains 5,000 pages, and each processor has 100 buffer pages of 4,000 bytes each. The cost of one page I/O is t_d, and the cost of shipping one page is t_s; tuples are shipped in units of one page by waiting for a page to be filled before sending a message from processor i to processor j. There are no indexes, and all joins that are local to a processor are carried out using a sort-merge join. Assume that the relations are initially partitioned using a round-robin algorithm and that there are 10 processors. Describe the evaluation plan, briefly, to find the highest paid employee and give its cost in terms of t_d and t_s. You should also compute the 'elapsed time' cost (i.e., if several operations are carried out concurrently, the time taken is the maximum over these operations).	**4**

Solution:

The round-robin partitioning implies that every tuple has an equal probability of residing at each processor. Moreover, since the sal field of Employees and budget field of Departments are uniformly distributed on 0 to 1,000,000, each processor must also have a uniform distribution on this range. Also note that processing a partial page incurs the same cost as processing an entire page and the cost of writing out the result is uniformly ignored. Finally, recall that elapsed time is the maximum time taken for any one processor to complete its task.

Plan: Conduct a complete linear scan of the Employees relation at each processor retaining only the tuple with the highest value in sal field. All processors except one then send their result to a chosen processor which selects the tuple with the highest value of sal. **[1.5]**

Total Cost $= (\#\text{ CPUs}) * (\text{Emp pg /CPU}) * (\text{I/O cost}) + (\#\text{ CPUs} - 1) * (\text{send cost})$

$= (10 * 10{,}000 * td) + (9 * ts)$

$= 100{,}000 * td + 9 * ts$ **[1.5]**

Elapsed Time $= 10{,}000 * td + ts$ **[1]**

Q. No.	Question	Marks
8. (b)	Consider a relation 'r' over the attributes A, B, C with the following characteristics: • 5,000 tuples with 5 tuples per page • Attribute A is a candidate key • Unclustered hash index on attribute A (assume that the cost involve in searching index is 2 I/Os) • Clustered B+ tree index on attribute B • Attribute B has 1,000 distinct values in r • Attribute C has 500 distinct values and an unclustered 3-level B+ tree index (i) Estimate the cost of computing $\sigma_{A=constant}(r)$ using the index. (ii) Estimate the cost of computing $\sigma_{B=constant}(r)$ using the index.	**6**

Solution:

(i) Since the hash index is on the candidate key, $\sigma_{A=constant}(r)$ has at most one tuple. Therefore, the cost is 2 (searching the index) + 1(retrieving data). If the index is integrated then the cost is just 2. **[3]**

(ii) Since B has 1,000 distinct values in r, there are about 5 tuples per value. Therefore $\sigma_{B=constant}(r)$ is likely to retrieve 5 tuples. Because the index is clustered and because there are 5 tuples per page, the result fits in 1 page. Therefore, the cost is depth of the tree + 1. **[3]**

Q. No.	Question	Marks
9. (a)	Write a MongoDB statement for each of the following SQL statements (Assume the name of database as 'mydb', name of collection as 'mycollection' and schema of R is (A, B, C, D,)):	**4**

(i) Select A, B, C From R Where D = 'p' or C = 'q';	(ii) Update R Set D = 'm' Where C = 'q' and A = 'n';
(iii) Select * From R Where A between 3 and 10;	(iv) Select A, C From R Where D in ('p', 'q', 'r');

Solution:

Sample Query:

(i) > db.mycollection.find({$or: [{D: 'p'}, {C: 'q'}]}, {A: 1, B: 1, C:1}) **[1]**

(ii) > db.mycollection.update({$and: [{C: 'q'}, {A: 'n'}]}, {$set: {D: 'm'}}) **[1]**

(iii) > db.mycollection.find({A: {$gte: 3, $lte: 10}}) **[1]**

(iv) > db.mycollection.find({D: {$in: ['p', 'q', 'r']}}, {A: 1, C:1}) **[1]**

Q. No.	Question	Marks
9. (b)	Write a Cassandra statement for each of the following: (i) Create a table (mytable) having - Attributes: A, B, C, D, & E (Assume suitable data type for each attributes). - Partition Keys: A & B - Cluster keys: D in ascending order and E in descending order. (ii) Write a CQL statement to retrieve all records of above created table whose attribute E value is 'p'.	**6**

Solution:

(i)
```
CREATE TABLE mytable
{
    A       type1,
    B       type1,
    C       type2,
    D       type3,
    E       type2,
    Primary Key ((A, B), D, E)
}
WITH CLUSTERING ORDER BY (D ASC, E DESC);
```
[4]

(ii)
```
Select *
From mytable
Where E = 'p' Allow filtering;
```
[2]

OR

```
CREATE INDEX ON mytable(E);
```
[2]

```
Select *
From mytable
Where E = 'p';
```