answer – 9.1

1605023

Here, relations are —

student(id, name, cgpa, yearAdmit)
Takes(id, course-id, semester, year)

and the query to be processed is —

select r.id, course-id, cgpa
from student r, takes s
where r.yearAdmit = s.year

Here, shared-nothing architecture is used. 10 nodes $n_1, n_2, ---, n_{10}$ are used to store r and s relations by horizontal partition vector on

id = 121000, 131000, 141000, 151000, 161000, 171000, 181000, 191000, 201000.

Date :

So, there are 10 partitions in total into 10 given nodes.

r is horizontally partitioned into $r_1, r_2, ---, r_{10}$.

s is horizontally partitioned into $s_1, s_2, ---, s_{10}$.

Here, the range of yearAdmit is 2015 to 2021.

a.

Here are total 7 values for yearAdmit/year in the given r and s relations. So, one of the partitions will contain two year values depending on data distribution. The 6 partitions of student and takes using yearAdmit are as follows -

i. r_1' and s_1' ⟶ yearAdmit/year < 2016 (v_1)

ii. r_2' and s_2' ⟶ 2016 (v_1) ≤ yearAdmit/year < 2017 (v_2)

iii. r_3' and s_3' ⟶ 2017 (v_2) ≤ yearAdmit/year < 2018 (v_3)

iv. r_4' and s_4' ⟶ 2018 (v_3) ≤ yearAdmit/year < 2019 (v_4)

v. r_5' and s_5' ⟶ 2019 (v_4) ≤ yearAdmit/year < 2020 (v_5)

vi. r_6' and s_6' ⟶ 2020 (v_5) ≤ yearAdmit/year.

b.

performing the given query with aforementioned 6 nodes -

steps

i. Repartition $r_1, ---, r_{10}$ into $r_1', ---, r_6'$. Do the same for $s_1, ---, s_{10}$ to get $s_1', ---, s_6'$. (based on year.)

ii. assign r_i', s_i' partitions to node N_i.

iii. perform $r_i' \bowtie s_i'$ at node N_i.

Ans.

answer – q.2

1605023

We have the following relations –

i. Customer (id, name, type, country)

ii. Purchase (id, product-id, p-country, date)

We have to process the following query –

Select r.id, product-id from customer r, purchase s where r.id = s.id

Here,

Shared-nothing architecture is used with nodes $N_1, \ldots, N_5$ where r and s are stored by horizontal partition vector on id = [1000, 2000, 3000, 4000]. So, we have 5 partitions in total into 5 nodes where –

i. r is horizontally partitioned into $r_1, \ldots, r_5$.

ii. s is horizontally partitioned into $s_1, \ldots, s_5$.

a. performing $r \bowtie_{r.id = s.id} s$ using provided 5 nodes –

steps

i. no repartitioning is required here.

ii. perform $r_i \bowtie s_i$ at Node N_i.

b.

Here, relations r and s are partitioned based on id and query attribute is also id. So, no repartitioning is required.

Ans.

answer-10.1 | 1605023

• problem on External Sort-merge (sort merge).

Here, in the given example,
number of runs, $N=4$ and number of memory blocks, $M=3$.

a

We need to find the size of the memory M' when the external sort (merge-sort) can be done in single pass.

In this case, as we have $N=4$ runs (each containing 3 tuples) after run creation step, we will need a memory with size $M'=5$ to accomplish the merge in one pass.
Then, 4 blocks of memory will be used as input buffer where each run will be assigned one memory block. The remaining block will serve as output buffer.
Again, if $M'=5$, then we will have 3 runs each containing 5 tuples except for the last run. A run with blank slots does not create any problem. It will be considered as one run in the merging step. $\boxed{N' = 12/5 \approx 3.}$

b. If the entire relation can be brought to memory simultaneously by increasing memory size, then we can apply any in-memory sorting algo. like quick sort on the relation. Usually, relations in DB are stored in disk for its size being huge. For sorting, we have to fetch a part of relation to memory from disk at a time. Hence, we can not apply in-memory sorting algo. directly on a relation.

So, memory size $M' =$ size of relation.

c. The number of merge passes required depends largely on the memory size. If we can complete the sorting with fewer passes, then required time will be reduced. Thus, the sorting performance improves and becomes efficient.

Ans.

answer – 10.2

| 1605023 |

given relation, Person (NID, name, DOB, street, city).
Here, the relation has 160 million tuples and is range-partitioned into 160 nodes using NID.
The following queries are to be processed:
a. find the list of persons sorted by NID in ascending order.
b. find the list of persons sorted by DOB in ascending order.

a. explanation of query processing

The tuples are range-partitioned based on sorting attribute (NID). So, tuples can be sorted locally in each node and concatenate the result to get output.

b. explanation of query processing

As the given relation has been partitioned based on attribute (NID) other than sorting attr. (DoB), we can sort it in two ways –

i. range – partitioning sort on DoB

ii. parallel external sort – merge algo.

Ans.

answer–11.1

1605023

Here, the 4 relations r_1, r_2, r_3, and r_4 are partitioned into 30 nodes $N_1, N_2, ---, N_{30}$.

We need to propose a query execution plan with intra-operation parallelism for individual joins and inter-operation pipeline parallelism.

a.

Here, the query is as follows –

$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 = \{(r_1 \bowtie r_2) \bowtie r_3\} \bowtie r_4 = (temp1 \bowtie r_3) \bowtie r_4 = temp2 \bowtie r_4$.

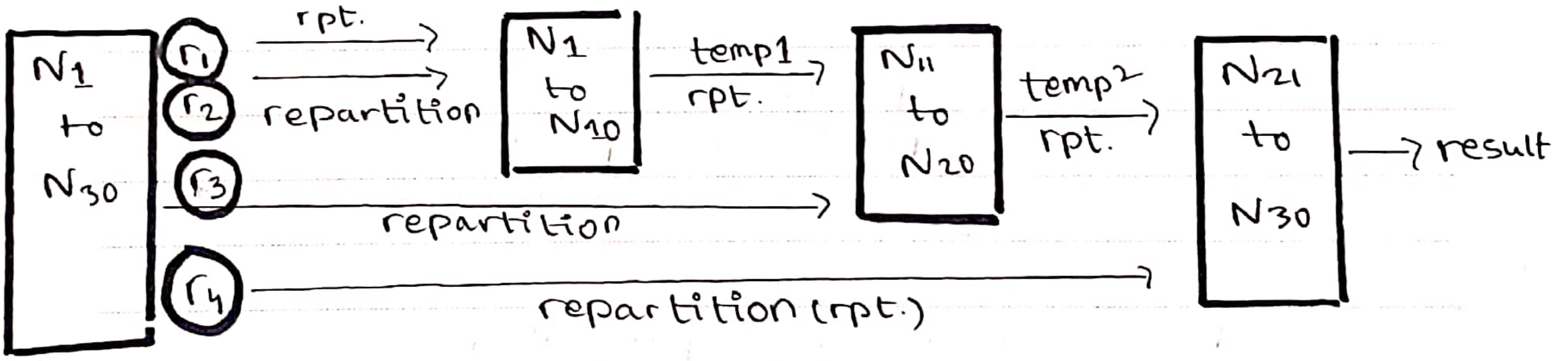
where,

- $r_1 \bowtie r_2 \longrightarrow$ intra – operation parallelism involved (individual joins)
- $(temp1 \bowtie r_3)$ & $(temp2 \bowtie r_4) \longrightarrow$ inter-operation pipelined parallelism involved.

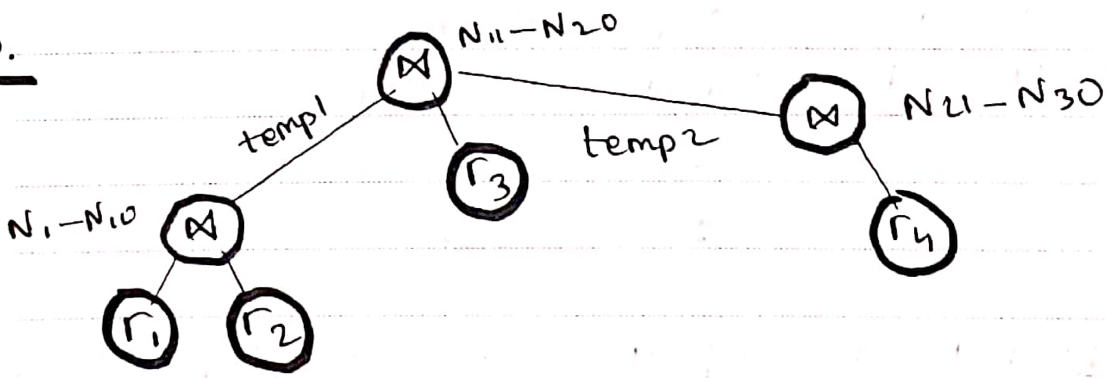
Here, among 30 nodes, $N_1 - N_{10}$ can be assigned to process the first joining ($r_1 \bowtie r_2$), $N_{11} - N_{20}$ can be assigned to process the second joining ($temp1 \bowtie r_3$), and the remaining 10 nodes can be assigned to process the third joining ($temp2 \bowtie r_4$).

So, we have to repartition r_1 and r_2 partitioned across 30 nodes to $N_1 - N_{10}$ based on query attribute. (intra-opt. parallelism)

We have to do the same repartitioning (based on query attribute) for pipelined output "temp1" and r_3 ($N_{11} - N_{20}$) as well as for pipelined output "temp2" and r_4 ($N_{21} - N_{30}$).



b.



Ans.

answer-11.2

$\boxed{1605023}$

a

we have to prepare query plan combining independent parallelism and pipelined parallelism for processing $S_1 \bowtie S_2 \bowtie S_3 \bowtie S_4 \bowtie S_5 \bowtie S_6$ where S_i $(1 \le i \le 6)$ = relations.

The nodes N_1, N_2, N_3, N_4, and N_5 are connected in a network and used to process the query.

Date :

Here, the joining will be independepent. for pairs of relations. Hence, we have the following operations —

i. $temp1 = r_1 \bowtie r_2$ (N_1)
ii. $temp2 = r_3 \bowtie r_4$ (N_2)
iii. $temp3 = r_5 \bowtie r_6$ (N_3)

} independent parallelism

iv. $temp4 = temp1 \bowtie temp2$ (N_4)
v. $result = temp3 \bowtie temp4$ (N_5)

} pipelined parallelism.

b.

independent parallelism	pipeline parallelism
i. less useful in highly parallel system	i. same
ii. operation has to pipelinable	ii. same
iii. limited pipeline (can not use all nodes)	iii. same
iv. supports all types of joining	iv. supports only a limited portion of joining types
v. joining operation can not start before inputs from previous step are fully fetched. Hence, it works slower considering this aspect.	v. pipelining allows joining to start before inputs from previous step are fully fetched. Hence, it works faster considering this aspect.

Ans.