End-term Exam (Fundamentals of Database Management System)-CSE202

Time: 1 hour 30 minutes
Total Marks: 30

Question-1: [5 Marks]

a) Consider a table recording how many hours students sleep at night:

Sleep (student, major, date, hours)

Assume the following for your answer:

A1: <*student, date*> is a key.

A2: Functional Dependency *student* **->** *major* holds.

Write a SQL query to find all students whose average night's sleep is shorter than the average over all students with the same major. **[3 Marks - Binary marking]**

Answer 1(a). Select student from Sleep as S1 group by student having avg(hours) < (Select avg(hours) from Sleep where major = S1.major)

b) Consider the following two SQL queries over table R(K,A) where K is a key.

Q1: *Select * from R where A >= all (select A from R)*

Q2: *Select * from R as R1 where A >= all (Select A from R as R2 where R1.K != R2.K)*

Are Q1 and Q2 equivalent? That is, do they return the same answer on all possible instances of R? If YES briefly justify why the queries are equivalent. If NO show the smallest instance of R that gives different answers for Q1 and Q2. **[2 Marks - Binary marking]**
[No marks will be awarded if proper justification is not given.]

Answer 1(b). Q1 and Q2 are equivalent.

Question-2:[5 Marks] Consider a relation R(A,B,C,D,E,F,G,H) with the following functional dependencies: **[No marks will be awarded if proper justification is not given.]**

A->BCD
AD-> E
EFG->H
F->GH

i) Based on these functional dependencies, determine all the keys for R. **[2 Marks - Binary marking]**
ii) One of the functional dependencies is redundant and can be removed without altering the keys. Which one is the redundant FD? **[2 Marks - Binary marking]**
iii) Is the relation in BCNF normal form? **[1 Mark - Binary marking]**

Answer 2(1). AF is the only key for relation R.

2(2). EFG -> H

2(3). No, relation R is not in BCNF.

Question-3: **[4 Marks]** For each of the schedule given below: **[4x1= Marks. 0.5 marks for recoverable and 0.5 for conflict serializable. No marks will be awarded if proper justification is not given.]**

- Indicate if the schedule is recoverable or not and
- Indicate if the schedule is conflict serializable or not.
- w denotes write operation, r denotes read operation, c denotes commit operation and a denotes an abort operation.

  a) S: w1(A), w2(B), w3(C), c1, r3(A), c2, r3(B), c3
  b) S: w1(A), w2(B), w3(C), r3(A), c2, r3(B), c3, c1
  c) S: r1(A), w2(A), w2(B), c2, r1(B), c1
  d) S: w1(A), w1(B), r2(A), w2(B), a2, w1(C),c1,r3(B), w3(C), c3

Answer 3(a). Recoverable, Conflict Serializable
3(b). Non-recoverable, Conflict Serializable
3(c). Recoverable, Not Conflict Serializable
3(d). Non-recoverable, Conflict Serializable

Question-4: [6 marks] Your database system is executing a 2-pass hash join with the following parameters: **[2 + 2 +2= Marks]**

- k: the number of buckets into which you will partition the relations.
- m: the total amount of main memory, in blocks.
- s: the number of blocks for each relation. (Assume that both the relations are of equal size.)

a) How large does *m* need to be so that the first phase of the algorithm can run?
b) How large does m need to be so that the second phase of the algorithm can run?
c) If the algorithm used is 3 pass hash-join instead of 2 pass hash-join, what will be the largest size of relations (both equal size) that can be joined?

Answer 4(a). First phase is to hash both R and S using the same hash function. Same attribute value in R and S will be hashed to same bucket. We can process the joining tuples by examining the corresponding pair of buckets alone. m>= k+1 as we need one buffer to bring a block of R or S in memory for hashing. **OR, $m > (s/m) +1 \Rightarrow m > \text{sqrt}(s)$.**

4(b). Second phase is to process each pair of the sub-relation Ri and Si using the one-pass join algorithm. The join attribute values are uniformly distributed. $m >= \text{ceil}(s/k) +1$

4(c). $m >= \text{ceil}(s/k^2) +1$ as the third pass re-hashes the relations into smaller buckets. $s<=mk^2$ for the three pass hash join.

Question-5: [10 marks] An outer-join of relations R and S is the union of the tuples from: (i) a regular natural join of R and S, (ii) the tuples from left relation R that did not match any from the right relation S, padded with NULLs, (iii) tuples from the right relation S that did not match any from the left relation R padded with NULLs. How can we adapt 2-pass sort merge-join algorithm used for performing natural join to compute outer-join without requiring any extra memory? Explain your algorithm through an example. Assume that relations R(A,B) and S(B,C) need to be joined. 2 tuples of R and S each can be stored in a block. There are only 4 memory buffers.

[Checking the condition that total number of blocks occupied by R and S in memory can be maximum three for the given configuration,, i.e. 24 tuples in total at max - 2 marks]

[Constraint that all tuples with single key of one relation must be completely in the memory - 2 marks]

[Correctly processing a corresponding blocks of sorted chunks of R and S to compute outer join -2 marks]

[Correctly discarding and bringing the next block of the processed sorted chunk - 2 marks]

[Giving a complete example which shows how blocks are brought in and removed from the memory -2 marks]

Answer.

```
If R.B == S.B
        Produce joining tuples at the output. R++
Else if R.B < S.B
        Produce the current tuple in R U Null at the output. R++
Else if S.B < R.B and S.B has not matched with any R.B before
        Produce the current tuple in S U Null at the output S++
```

Example: Consider the relations R and S below. There are only 4 memory buffers as mentioned in the question.

R

A	**B**
A1	B1
A1	B2
A2	B2
A2	B3
A3	B1
A3	B1
A4	B1
A5	B3
A6	B7
A7	B8
A9	B5

S

B	**C**
B1	C1
B2	C1
B3	C2
B10	C2

R outer join S

A	**B**	**C**
A1	B1	C1

A1	B2	C1
A2	B2	C1
A2	B3	C2
A3	B1	C1
A3	B1	C1
A4	B1	C1
A5	B3	C2
A6	B7	NULL
A7	B8	NULL
A9	B5	NULL
NULL	B10	C2

As the memory contains 4 buffers – 1 buffer will be used to store the output and 3 buffers can store chunks from R and S. We store 2 chunks of R and 1 chunk of S in the main memory.

R1 (FIRST SORTED CHUNK OF R)

B1	B1	B1	B1	B2	B2	B3	B3
A1	A3	A3	A4	A1	A2	A2	A5

R2 (SECOND SORTED CHUNK OF R)

B5	B7	B8					
A9	A6	A7					

S1 (FIRST SORTED CHUNK OF S)

B1	B2	B3	B10				
C1	C1	C2	C2				

MEMORY

B1 B1 (CHUNK FROM R1)

```
A1       A3
--------------
B5       B7       (CHUNK FROM R2)
A9       A6
--------------
B1       B2       (CHUNK FROM S1)
C1       C1
--------------
O/P
---------------
```
In the above snapshot of the memory, (B1, A1) and (B1, A3) from R1 will match with (B1,C1) from S1 and output tuples (A1,B1,C1) and (A3,B1,C1) will be produced.

MEMORY

```
B1       B1       (CHUNK FROM R1)
A3       A4
--------------
B5       B7       (CHUNK FROM R2)
A9       A6
--------------
B1       B2       (CHUNK FROM S1)
C1       C1
--------------
O/P
---------------
```
In the above snapshot of the memory, (B1, A3) and (B1, A4) from R1 will match with (B1,C1) from S1 and output tuples (A3,B1,C1) and (A4,B1,C1) will be produced.

MEMORY

```
B2       B2       (CHUNK FROM R1)
A1       A2
--------------
B5       B7       (CHUNK FROM R2)
A9       A6
--------------
B1       B2       (CHUNK FROM S1)
C1       C1
--------------
O/P
---------------
```

As (B1,C1) from S1 has been processed before, it will be removed from the memory. (B2, A1) and (B2, A2) will join with (B2,C1) from S1 and (A1,B2,C1), (A2,B2,C1) will output.

```
 MEMORY

B3      B3      (CHUNK FROM R1)
A2      A5
--------------
B5      B7      (CHUNK FROM R2)
A9      A6
--------------
B3      B10     (CHUNK FROM S1)
C2      C2
--------------
O/P
---------------
```

In the above snapshot of the memory, (B3, A2) and (B3, A5) from R1 will match with (B3,C2) from S1 and output tuples (A2,B3,C2) and (A5,B3,C2) will be produced. R1 will become empty now.

```
MEMORY

EMPTY           (CHUNK FROM R1)

--------------
B5      B7      (CHUNK FROM R2)
A9      A6
--------------
B3      B10     (CHUNK FROM S1)
C2      C2
--------------
```

O/P

As B5 < B10, (A9, B5, NULL) will be produced and pointer will move to (B7, A6) in R2. B7 < B10, (A6, B7, NULL) will be produced and next tuple from R2 will be fetched in memory.

MEMORY

```
EMPTY         (CHUNK FROM R1)

--------------
B8            (CHUNK FROM R2)
A7
--------------
B10           (CHUNK FROM S1)
C2
--------------
```

O/P

As B8 < B10, (A7, B8, NULL) will be produced and R2 will become empty. A tuple (NULL, B10, C2) will also be produced.