COMP 3311
DATABASE MANAGEMENT SYSTEMS

FINAL REVIEW

INDEXING

A. A page can hold either 3 records or 10 (search-key, pointer) index entries. If a database contains n records, then how many pages are needed to store both the data file and a single-level dense index?

a) n/30
b) 3n/10
c) 10n/3
d) 13n/30

number of records = n $bf_{\text{data file}}$ = 3 bf_{index} = 10

Pages needed for the data file: n/3

Pages needed for the single-level dense index: n/10

Total pages needed: n/3 + n/10 = 13n/30

INDEXING (CONT'D)

B. In a B^+-tree, if the search-key value is 12 bytes, the page size is 1024 bytes and a pointer is 6 bytes, then the maximum number of search-key values that can be stored in each non-leaf node of the B^+-tree is:

a) 54

b) 56

c) 57

d) 58

Each index entry is 12 + 6 = 18 bytes.

Therefore, $\lfloor 1024 / 18 \rfloor$ = 56 index entries (i.e., search-key values) can be stored in a non-leaf node.

INDEXING (CONT'D)

C. What is the minimum number of search-keys in any non-root node for a B^+-tree in which the maximum number of search-keys in a node is 4?

a) 1

b) 2

c) 3

d) 4

The fan out of the B^+-tree, n, is 5 (i.e., one more than the number of values).

For internal nodes, the minimum number of values is $[n/2]-1 = [5/2]-1 = 2$.

For leaf nodes, the minimum number of values is $[(n-1)/2] = [4/2] = 2$.

Therefore, for any non-root node the minimum number of values is 2.

EXTERNAL SORTING

The relation Sailor(sailorId, name, rating, age) is not sorted. Assume each attribute is 25 bytes, the page size is 1,000 bytes and there are 11,000 tuples. For the following questions, apply external sorting using a buffer of 11 pages.

A. How many sorted runs will be produced in pass 0?

a) 10
b) 11
c) 100
d) 110

tuple size: 100 bytes
bf_{Sailor}: 10 tuples/page
B_{Sailor}: 1100 pages
buffer M: 11 pages

In pass zero, 11 pages at a time are sorted in memory.

Therefore, in pass zero $\lceil 1100/11 \rceil$ = 100 sorted runs are produced.

EXTERNAL SORTING (CONT'D)

B. What is the total number of passes required to sort the relation completely (including pass 0)?

a) 2
b) 3
c) 4
d) 5

tuple size: 100 bytes
bf_{Sailor}: 10 tuples/page
B_{Sailor}: 1100 pages
buffer M: 11 pages

After pass zero there are 100 sorted runs.

In pass 1, 10 runs at a time are merged, producing 10 sorted runs.

In pass 2, these 10 sorted runs are merged to produce the final output.

Therefore, the total number of passes = 3.

EXTERNAL SORTING (CONT'D)

C. What is the total page I/O cost of sorting the relation?

a) 4,400
b) 5,500
c) 6,600
d) 7,770

tuple size: 100 bytes
bf_{Sailor}: 10 tuples/page
B_{Sailor}: 1100 pages
buffer M: 11 pages

There are a total of three passes.

In each pass the whole relation (1100 pages) is read and written.

Therefore, the total page I/O cost is 3 * 2 * 1100 = 6,600

QUERY PROCESSING

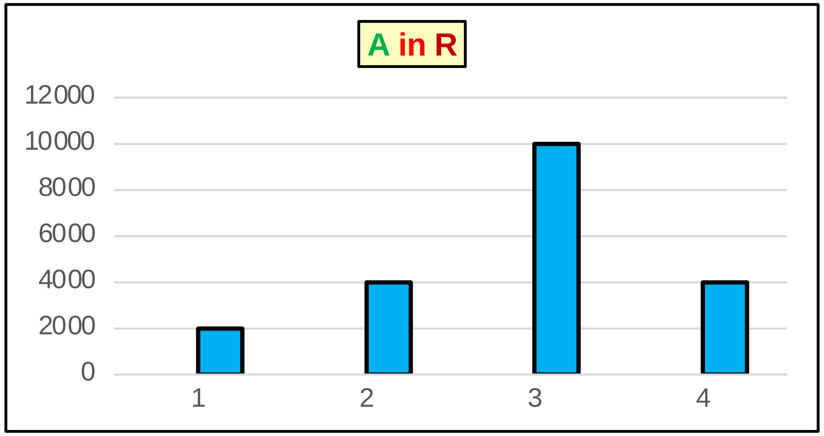
Consider the two relations R(*B, C*, A) and S(B, A, Y).

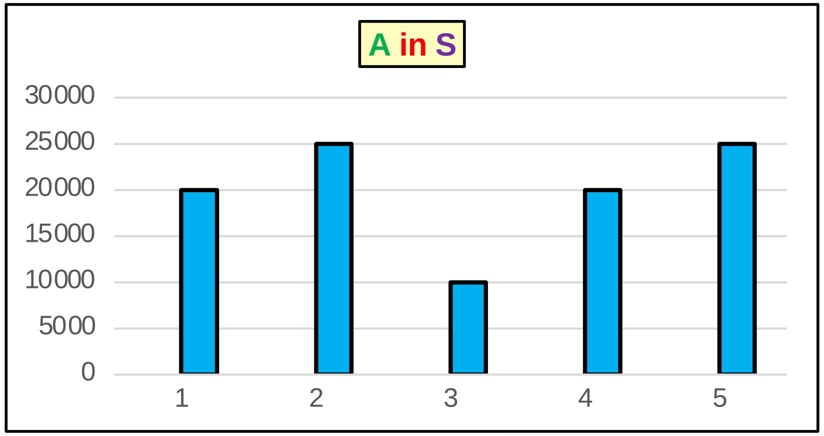
R contains 20,000 tuples and S contains 100,000 tuples.

Assume that for both relations, 10 tuples fit per page (i.e., the size of R is 2,000 pages and that of S is 10,000 pages).

The possible values of attribute A in R are {1, 2, 3, 4}, whereas the possible values of A in S are {1, 2, 3, 4, 5}.

The following histograms present statistical information about the occurrences of values for A in R and S (e.g., there are 2,000 tuples with A=1 in R and 20,000 tuples with A=1 in S).





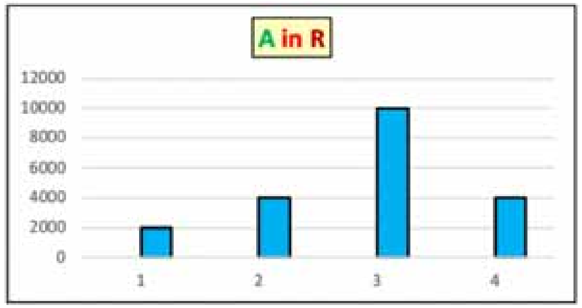
R(B, C, A)
S(B, A, Y)

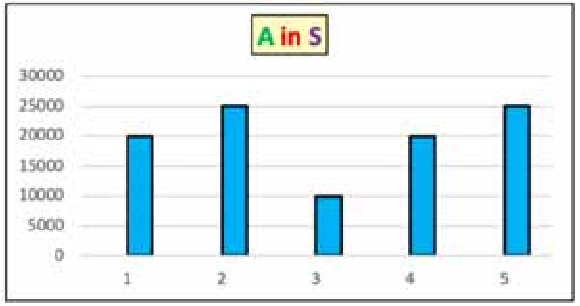
QUERY PROCESSING (CONTD)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

A. How many tuples are there in the query result of (R $\text{JOIN}_{R.A=S.A}$ S) (i.e., what is the cardinality of the output)?

a) 20,000
b) 100,000
c) 320,000,000
d) 400,000,000





A is not a key for R or S.

According to the histograms, each tuple of R with value R.A=1 (of which there are 2,000) can join with 20,000 tuples of S with S.A=1. That is, the join result will contain 2,000 * 20,000 = 40×10^6 tuples for R.A=S.A=1.

Performing the same computation for A equals 2, 3, 4 and 5 the result will contain $(40 + 100 + 100 + 80 + 0) \times 10^6$ = 320,000,000 tuples.

R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

B. What is the minimum page I/O cost to compute (R $\text{JOIN}_{R.A=S.A}$ S) using block nested-loop join; how many buffer pages are needed?

a) Minimum page I/O cost is 12,000; 2,000 buffer pages are needed.

b) Minimum page I/O cost is 12,000; 2,002 buffer pages are needed.

c) Minimum page I/O cost is 320,000 and 2,002 buffer pages are needed.

d) Minimum page I/O cost is 320,000; 12,000 buffer pages are needed.

Since there is no index, at least both relations must be read (i.e., the minimum page I/O cost is 2,000 + 10,000 = 12,000 *independent of the algorithm*.

For block nested-loop join, the smaller relation R (2,000 pages) needs to be able to be kept in memory, so that S is scanned only once.

Since 1 page is needed for reading S and 1 page for holding the output, 2,002 buffer pages are needed.

R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

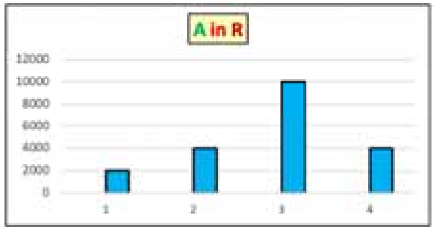
C. We want to compute (R $JOIN_{R.A=S.A}$ S) using block nested-loop join with R *as the outer relation*. What is the minimum number of buffer pages needed in order to achieve a page I/O cost of 42,000?

a) 502
b) 2,000
c) 2,002
d) 10,002

Since R (2,000 pages) is the outer relation and the total page I/O cost is 42,000, S needs to be scanned (42,000 - 2,000) / 10,000 = 4 times.

If S is scanned 4 times, this means that 4 "blocks" of R need to be read to join with S, and each "block" must be at least 2,000 / 4 = 500 pages.

Since, 1 page is needed for reading S and 1 page for the output, in total 502 buffer pages are needed.



R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

D. We want to compute (R $JOIN_{R.A=S.A}$ S) using hash join with R *as the build input*. How many buckets should be used for partitioning and what is the minimum buffer requirement?

a) 4 buckets; 202 buffer pages.
b) 4 buckets; 1,002 buffer pages.
c) 5 buckets; 202 buffer pages.
d) 5 buckets; 102 buffer pages.

The join condition is R.A=S.A and there are only four values of A in R. Therefore, 4 buckets should be used.

According to the R histogram, 10,000 tuples of the build input A, have value R.A=3. Thus, the bucket size should be ⌈10,000 / 10⌉ = 1,000 pages (*bf*=10) so that each bucket of R can fit into the buffer.

In addition, we need 1 page for reading S and 1 for the output.

Therefore, 1,002 buffer pages are needed.

R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

E. Given that R.B is a not null foreign key referencing S.B, how many tuples are in the query result of (R $\text{JOIN}_{R.B=S.B}$ S)?

a) 20,000
b) 100,000
c) 120,000
d) 320,000

Each tuple of R joins with exactly one tuple in S.

Therefore, the size of the join result is the same as the size of R, namely, 20,000 tuples.

R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

F. We want to compute (R $\text{JOIN}_{R.B=S.B}$ S) using indexed nested-loop join with R *as the outer relation*. Assume that there is a hash index on S.B with no overflow buckets (i.e., finding an index entry has page I/O cost 1). What is the join page I/O cost?

a) 6,000

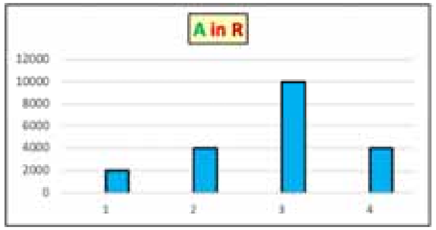
b) 12,000

c) 22,000

d) 42,000

For each tuple of R, the hash index entry with the corresponding value of B is found and the pointer followed to the tuple of S for a page I/O cost of 2 per tuple. This is repeated for all 20,000 tuples of R, for a total page I/O cost of 20,000 * 2 = 40,000.

In addition, the 2,000 pages of R must be read for a total page I/O cost of 40,000 + 2,000 = 42,000.



R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONTD)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

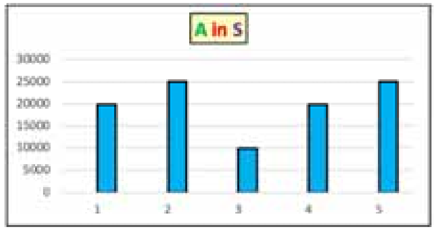
G. How many tuples are there in the result of (($\sigma_{A=1}$R) JOIN$_{R.B=S.B}$ S) and what is the minimum query processing page I/O cost using indexed nested-loop join with R *as the outer relation*. Assume that the only index is a hash index on S.B with no overflow buckets.

a) The result has 2,000 tuples; the page I/O cost is 6,000.
b) The result has 2,000 tuples; the page I/O cost is 22,000.
c) The result has 20,000 tuples; the page I/O cost is 40,000.
d) The result has 20,000 tuples; the page I/O cost is 42,000.

According to the R histogram, there are only 2,000 tuples in R with R.A=1. Each of these tuples, matches exactly 1 tuple in the join with S (R.B=S.B). Thus, the result contains 2,000 tuples.

Finding the matching S tuple has page I/O cost 2 (see previous question) and so finding all matches for the 2,000 tuples has page I/O cost 4,000.

Adding the cost of reading R (2,000 page I/Os), the minimum query processing page I/O cost is 6,000.



R(B, C, A)
S(B, A, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

H. How many tuples are there in the result of (($\sigma_{A=1}$R) JOIN$_{R.B=S.B}$ ($\sigma_{A=3}$S)) and what is the minimum query processing page I/O cost using index nested-loop join with R *as the outer relation*. Assume that the only index is a hash index on S.B with no overflow buckets.

a) The result has 200 tuples; the page I/O cost is 600.
b) The result has 200 tuples; the page I/O cost is 6,000.
c) The result has 2,000 tuples; the page I/O cost is 6,000.
d) The result has 2,000 tuples; the page I/O cost is 42,000.

According to the S histogram, among the 2,000 tuples in the result of (($\sigma_{A=1}$R) JOIN$_{R.B=S.B}$ S), only 10,000 / 100,000 = 0.1 (i.e.,10%) are expected to satisfy the additional condition S.A=3. Thus, the result is expected to contain 2,000 * 0.1 = 200 tuples.

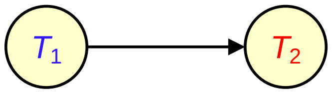
The minimum query processing page I/O cost is 6,000, the same as in the previous question.

TRANSACTION MANAGEMENT

4. Consider the following schedules of two transactions T_1 and T_2. Indicate for each whether it is serial, (conflict) serializable or not serializable. r denotes a read and w a write operation.

   a) Schedule: $r_1(A)\ w_1(A)\ r_2(A)\ w_2(B)$

   Serial: $T_1\ T_2$

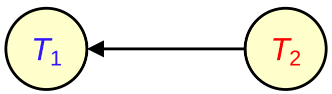


T_1	T_2
read(A)	
write(A)	
	read(A)
	write(B)

4. Consider the following schedules of two transactions T_1 and T_2. Indicate for each whether it is serial, (conflict) serializable or not serializable. r denotes a read and w a write operation.

   b) Schedule: $r_1(A)\ r_2(A)\ w_1(A)\ w_2(B)$

   Serializable: $T_2\ T_1$

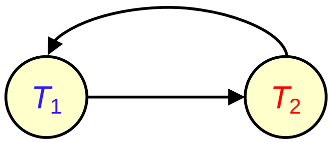


T_1	T_2
read(A)	
	read(A)
write(A)	
	write(B)

4. Consider the following schedules of two transactions T_1 and T_2. Indicate for each whether it is serial, (conflict) serializable or not serializable. r denotes a read and w a write operation.

   c) Schedule: $r_1(A)$ $r_2(A)$ $w_2(A)$ $w_1(A)$

   Not Serializable



T_1	T_2
read(A)	
	read(A)
	write(A)
write(A)	

5. Consider the schedule $r_1(A)$ $w_1(A)$ $r_2(A)$ $w_2(B)$ c_1 c_2 (where c_1 and c_2 indicate the commit statements).

   a) Is the schedule recoverable? Why?

   It is recoverable (T_2 reads database item A written by T_1, but T_1 commits before T_2).

   Is the schedule cascadeless? Why?

   It is not cascadeless (T_2 read database item A before T_1 commits).

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(B)
commit	
	commit

5. Consider the schedule $r_1(A)\ w_1(A)\ r_2(A)\ w_2(B)\ c_1\ c_2$ (where c_1 and c_2 indicate the commit statements).

   b) Change the time of the commits (c_1, c_2) in the schedule in a) so that it becomes a cascadeless schedule.

      $r_1(A)\ w_1(A)\ c_1\ r_2(A)\ w_2(B)\ c_2$

      Transaction T_2 reads the value of A written by T_1, <u>after</u> T_1 commits.

T_1	T_2
read(A)	
write(A)	
commit	
	read(A)
	write(B)
	commit

TRANSACTION MANAGEMENT (CONT'D)

5. Consider the schedule $r_2(A)$ $r_1(A)$ $w_1(A)$ $w_2(B)$ c_2 c_1 (where c_1 and c_2 indicate the commit statements).

   (c) Is the schedule $r_2(A)$ $r_1(A)$ $w_1(A)$ $w_2(B)$ c_2 c_1 recoverable and cascadeless?

   Both recoverable and cascadeless – no transaction reads items after they have been written by the other.

T_1	T_2
	read(A)
read(A)	
write(A)	
	write(B)
	commit
commit	

TRANSACTION MANAGEMENT (CONT'D)

6. Modify the schedule $r_2(A)$ $r_1(A)$ $w_1(A)$ $w_2(B)$ according to the 2PL protocol by adding lock-s, lock-x, unlock statements. Explain briefly whether the schedule is allowed by 2PL.

   The schedule is allowed by 2PL . T_1 must wait until T_2 unlocks A.

T_1	T_2
	lock-s(A)
	read(A)
lock-s(A)	
read(A)	
lock-x(A) ⟹ **wait**	
write(A)	
	write(B)

7. Modify the schedule r_2(A) r_1(A) w_1(A) w_2(B) according to timestamp-ordering protocol by adding RTS (read timestamp) and WTS (write timestamp) statements. Assume that the timestamps of T_1 and T_2 are 2 and 1, respectively. The initial read and write timestamps of A and B are both 0.

T_1 [TS=2]	T_2 [TS=1]
	read(A) RTS(A)=1; WTS(A)=0
read(A) RTS(A)=2; WTS(A)=0	
write(A) RTS(A)=2; WTS(A)=2	
	write(B) RTS(B)=0; WTS(B)=1

Read
If TS(T_i) < WTS(Q) **rollback**
If TS(T_i) ≥ WTS(Q)
RTS(Q) = max(TS(T_i), RTS(Q))

Write
If TS(T_i) < RTS(Q) **rollback**
If TS(T_i) < WTS(Q) **ignore**
Otherwise WTS(Q) = TS(T_i)

8. Modify the schedule $r_2(A)$ $r_1(A)$ $w_1(A)$ $w_2(B)$ according to the multi-version timestamp-ordering protocol by adding RTS (read timestamp) and WTS (write timestamp) statements and specify the versions of the items. Assume that the timestamps of T_1 and T_2 are 1 and 2, respectively and that the initial versions of items are A_0 and B_0 with values 0 for their read and write timestamps. Complete the correct version numbers (e.g., read(A_0) instead of read(A)).

T_1 [TS=1]	T_2 [TS=2]
	read(A_0) RTS(A_0)=2; WTS(A_0)=0
read(A_0) RTS(A_0)=2; WTS(A_0)=0	
write(A) TS(T_1)=1 < RTS(A_0)=2 ⟹ **rollback**	
	write(B_1) RTS(B_1)=2; WTS(B_1)=2

Read

Reads always succeed

set RTS(Q_k) = max(TS(T_i), RTS(Q_k))

Write

If TS(T_i) < RTS(Q_k) **rollback**

If TS(T_i) = WTS(Q_k)

overwrite contents

If TS(T_i) > WTS(Q_k)

create new version

set R/WTS(Q')=TS(T_i)