SWE3003 Database Systems - Final Exam Fall 2020

- **[Note 1] Write down your student ID and name on top of each answer sheet.**
- **[Note 2] You are not allowed to leave WebEx meeting until the exam is over.**
- **[Note 3] No questions are allowed. If you find a question is not clear and needs some assumptions, you should make and write down your own assumptions and solve it based on them.**
- **[Note 4] For the scheduling problems, you should assume each transaction starts with the first operation shown in the tables.**

1. [Storage and File Structures] You have a database table that consists of 20,000 records. Each 4 KB disk page holds 20 records without free space. I.e., at least 1,000 disk pages are required for the database table. Suppose your DBMS does not use the buffer cache. For each of the following database table format, describe how many disk pages need to be accessed on average for each insert or point (exact match) query.

   (a) Heap file - insert

   1 page

   (b) Heap file - search

   average, 500 pages

   (c) Sequential file - insert

   $log_2 1000$ + 500 pages

   (d) Sequential file - search

   $log_2 1000$ pages

   (e) B+tree file - insert

   $log_{100} 1000$ + 1 pages

   (f) B+tree file - search

   $log_{100} 1000$ pages

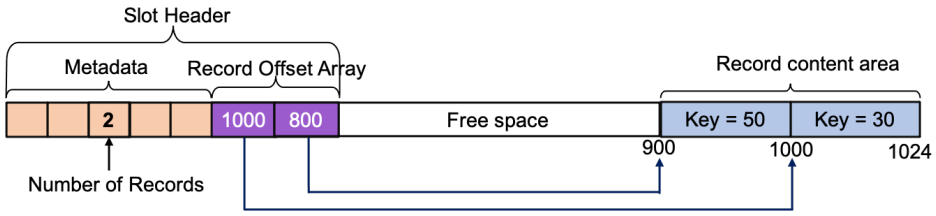
2. [Storage and File Structures] The following figure shows a slotted page that holds two records of variable length. How will this page be changed if we execute the following sequence of operations. Draw the page for each step.

   (a) Query 1: We insert a record of 50 Bytes whose key is 40 into the page. How will the page be changed?

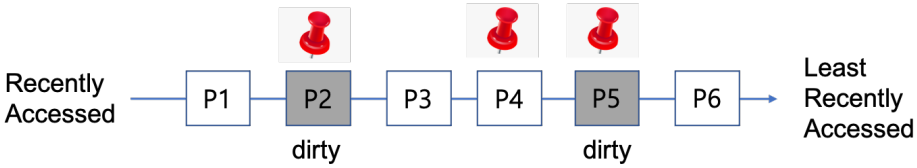
3, 1000, 850, 900.... 850(key=40), 900(key=50), 1000(key=30)

(b) Query 2: Then, we delete the existing record whose key is 50 from the page that Query 1 modified. How will the page be changed?

2, 1000, 850 .... 850(key=40), hole, 1000(key=30)



3. [Buffer Cache] Suppose a buffer manager can hold up to six pages in a DBMS that we designed, as shown in the figure below. We decided to use the LRU replacement policy for the buffer manager. In the figure, the dark colored pages are dirty, and the white colored pages are clean. The red pins indicate which pages are pinned (currently accessed) by a transaction. In the example, a suspended transaction T0 is accessing P2, P4, and P5.



(a) Suppose the suspended transaction T0 has not waken up. If a new read-only transaction T1 reads page P6 and and P7 when the buffer manager's LRU list is as shown in the figure, how many disk pages will be flushed to disks? Draw the status of LRU list after the transaction T1 commits.

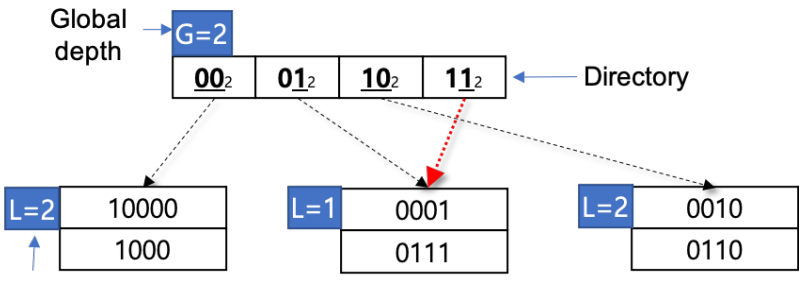
0 pages will be flushed.
P7 → P6 → P1 → P2 (dirty, pinned) → P4 (pinned) → P5 (dirty, pinned)

(b) The suspended transaction T0 that pinned P2, P4, and P5 has just woken up when the buffer manager's LRU list is as shown in the figure. If the transaction T0 commits, how many disk pages will be flushed to disks? Draw the status of LRU list after the transaction T0 commits. If necessary, you can make assumptions about the buffer cache management policies.

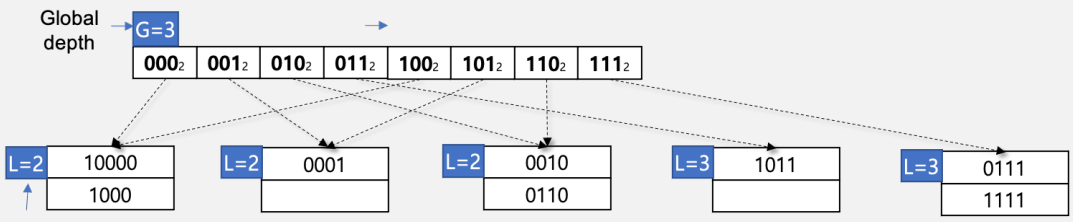
2 pages will be flushed.
P1 → P2 → P3 → P4 → P5 → P6 (all pages are clean and unpinned)
or, P1 → P3 → P4 → P6 (all pages are clean and unpinned)

4. [Extendable Hashing] Suppose we have the following extendable hash table. We use the least significant bits of keys and $\%2^G$ as the hash function. If each bucket can hold up to 2 records, how will the extendable hash table change if we insert the following records? Please draw an extendable hash table.

Record	Hash key
r15	1111
r11	1011

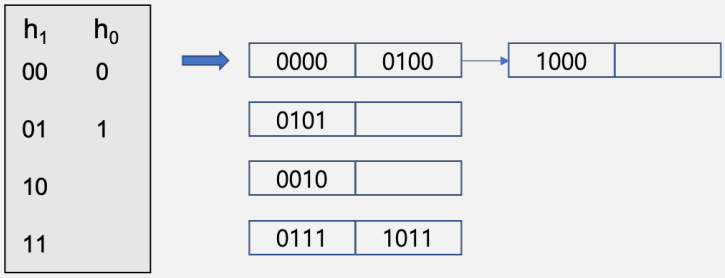


Solution:



5. [Linear Hashing] Suppose we insert the following records into an empty linear hash table. The initial number of buckets (N) is 2 when the level (L) is 0. I.e., the hash function family is $\%2^L \times N$. If each bucket can hold up to 2 records, how will the linear hash table change after all the following records are inserted? Please draw a linear hash table that holds the following records.

Record	Hash key
r0	0000
r2	0010
r4	0100
r5	0101
r7	0111
r8	1000
r11	1011



6. [Indexing] Suppose you have a table that has a very large number of attributes and a millions of records. If your application runs the following type of queries frequently, what type of index on which attributes will help improve the query processing performance? Justify your answer.

```
SELECT attr1
FROM myTable
WHERE attr2 = val2 AND attr3 = val3 AND attr4 > val4 AND attr4 < val5;
```

attr2: Hashing
attr3: Hashing
attr4: B+tree

7. [Join] Suppose you have two tables - `student` and `department`. `student` table has 1,000,000 records and `department` table has 100,000 records. Each disk page can hold maximum 10 student records or 10 department records.

   (a) How many passes are required for external merge sort for each table if your DBMS can use 5 buffer pages?

   (a) department table has 100,000 pages, and student table has 10,000 pages. The initial number of runs is ceil(b/M), i.e., 20,000 and 2,000, respectively. Then, the number of runs decreases by a factor of 4.
   I.e., → 2nd pass: 5,000 and 500,
   → 3rd pass: 1,250 and 125
   → 4th pass: 313 and 32,
   → 5th pass: 79 and 8
   → 6th pass: 20 and 2
   → 7th pass: 5
   → 8th pass: 2
   Therefore, the answers are 8 and 6.

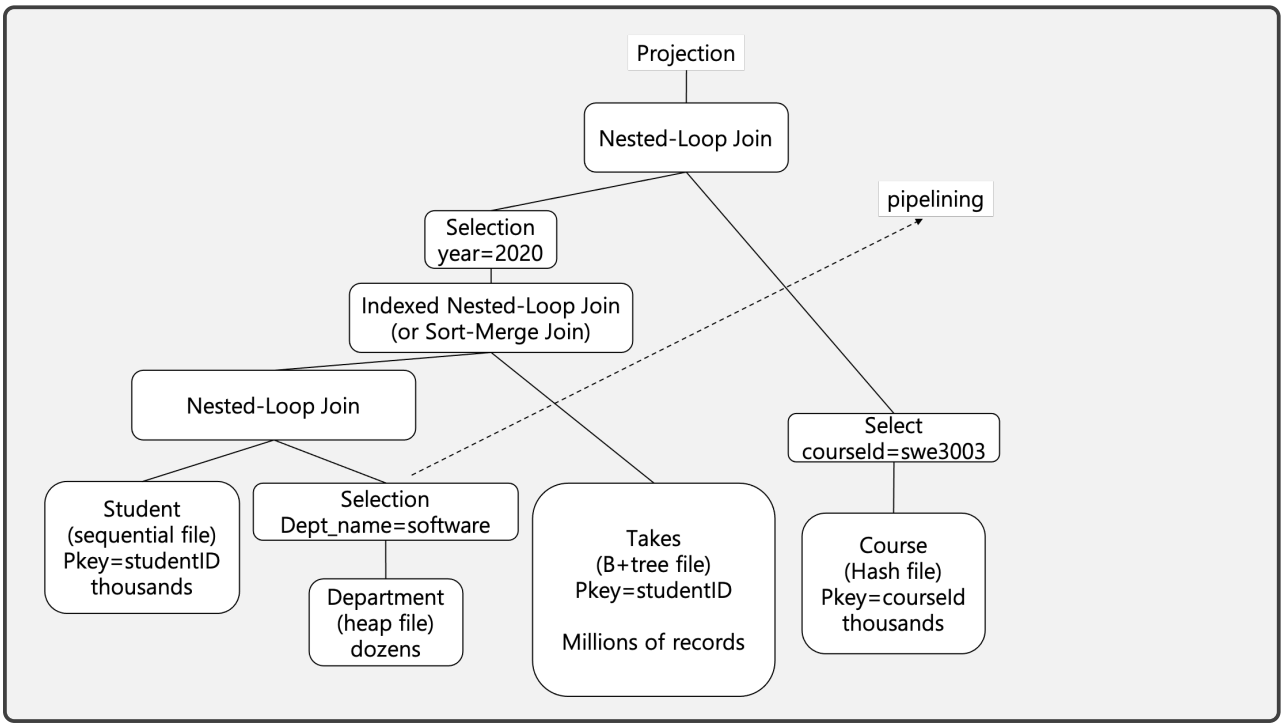
   (b) What is the expected number of disk accesses if we employ Sort-Merge Join using 4 input buffer pages and 1 output buffer page? Show how you reached the answer.

   The total number of block transfers for external sorting = $b(2\lceil log_{M-1}(b/M)\rceil + 1)$
   For department table, $100,000 * (2 * \lceil log_4 25,000\rceil + 1) = 100{,}000 * (2 * 8 + 1) = 1{,}700{,}000$
   For student table, $10,000 * (2 * \lceil log_4 2,500\rceil + 1) = 10{,}000 * (2 * 6 + 1) = 130{,}000$
   For join, we scan 100,000 + 10,000 pages.
   Therefore, total number of block transfers is 1,830,000 + 110,000 = 1,940,000 pages.

8. [Query Optimization] Suppose you are have `student`, `department`, `takes`, and `course` tables. `student` table has thousands of records and it is stored in a Sequential file. Its primary key is the student ID. `department` table has dozens of records and it is stored in a heap file without a secondary index. `takes` table has millions of records in a B+tree file. Its primary key is student ID and there is no secondary index for course ID. `course` table has thousands of records stored in a hash table, where course ID is used as the hash key. Consider the following query.

```
SELECT student.name
FROM student JOIN department JOIN takes JOIN course
WHERE course.course_id = 'swe3003' AND takes.year = 2020
      AND department.dept_name = 'software';
```

   Draw an expression tree for a query execution plan that you think the most efficient. Explain why your query plan will be efficient.



9. [Serializability] Consider the schedule shown in Figure 1 where time increases from top to bottom.

T1	T2	T3
		R(B)
R(A)		
	R(B)	
W(B)		
	R(A)	
		W(A)
Commit		
	Commit	
		Commit

Figure 1: Schedule 1

(a) Is this schedule conflict serializable? Justify your answer.

No, it is not.
T1: R(A) → T3: W(A)
T3: R(B) → T1: W(B)
are conflicting, and they can not be reordered.
I.e., this schedule has cyclic dependency.
Therefore, this schedule cannot be transformed into a serial schedule.

(b) Is this schedule view serializable? Justify your answer.

No, it is not.

(answer 1) Every view serializable schedule that is not conflict serializable has blind writes. But, this schedule is not conflict serializable and it does not have blind writes.
(answer 2) For A, T1 performs initial read, and T3 should write it. For B, T3 must perform initial read, and T1 should write it later. But, due to the "cyclic dependency", We cannot satisfy both conditions.
Grading: if students wrote something about view serializability definition, please give some partial credits.

10. [2PL] Consider the schedule shown in Figure 2. This schedule is not a conflict-serializable. Show how 2PL (Two Phase Locking) can ensure a conflict-serializable schedule for the transactions. Use the notation L(A) to indicate that the transaction acquires the lock on element A and U(A) to indicate that the transaction releases its lock on A.

T1	T2
R(A)	
W(A)	
	R(A)
	R(B)
	Commit
R(B)	
W(B)	
Commit	

Figure 2: Schedule 2

Solution:

T1	T2
L(A)	
R(A)	
W(A)	
	L(A) → block
L(B)	
R(B)	
W(B)	
U(A)	
U(B)	
Commit	
	L(A) → granted
	R(A)
	L(B)
	R(B)
	U(A)
	U(B)
	Commit

11. [Multiple granularity locking] Consider a database system holding the records shown below that implements hierarchical multiple granularity locking protocol. Suppose transaction T0 acquired various locks on tree nodes using multiple granularity protocol as shown in Figure 3. For each of the following transactions, indicate what kind of locks (SIX, IX, X, IS, S) on which tree node it can acquire or not. Justify your answer.

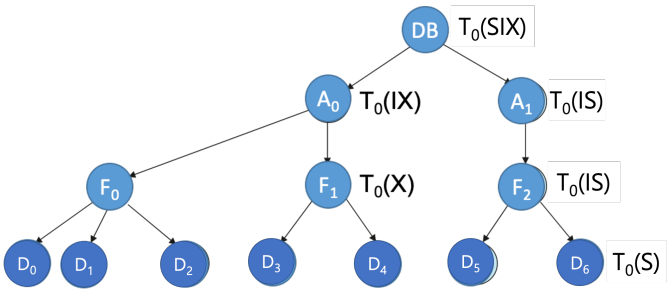


Figure 3: Granularity Hierarchy

(a) Write(D5)

This transaction tries to acquire IX lock on DB first, but IX is not compatible with SIX. So, it is blocked in the root node.

(b) Read(D3)

This transaction tries to acquire the following locks.
DB: IS (compatible with SIX)
A0: IS (compatible with IX)
F1: IS (not compatible with X), so it is blocked in F1.

(c) Read (F1)

This transaction tries to acquire the following locks.
DB: IS (compatible with SIX)
A0: IS (compatible with IX)
F1: S (not compatible with X), so it is blocked in F1.

(d) Add a new tree node F3 under A1.

This transaction tries to acquire the following locks.
DB: IX (not compatible with SIX), so it is blocked in DB.

(e) Read(DB) and Write(F2)

This transaction tries to acquire the following locks.
DB: S (not compatible with SIX)
DB: IX (also not compatible with SIX)
so, it is blocked in the root level.

12. [Timestamp-ordering] Consider the following schedule under timestamp-ordering protocol. Initially, the write and read timestamps of data `A` and `B` are all 0, and the timestamps of `T1` and `T2` are 1 and 2, respectively.

T1	T2
R(A)	
	R(A)
	W(A)
	R(B)
W(B)	
Display(A+B)	
	W(B)
	Commit
Commit	

Figure 4: Timestamp Ordering Protocol

(a) Is the schedule valid under timestamp-ordering protocol? Justify your answer.

No. This is not a valid schedule.

- T1: R(A) $\rightarrow$ OK because TS(T1)=1 > W-timestamp(A)=0 and TS(T1)=1 > R-timestamp(A)=0. $\therefore$ R-timestamp(A) $\leftarrow$ 1
- T2: R(A) $\rightarrow$ OK because TS(T2)=2 > W-timestamp(A)=0 and TS(T1)=1 > R-timestamp(A)=1. $\therefore$ R-timestamp(A) $\leftarrow$ 2
- T2: W(A) $\rightarrow$ OK because TS(T2)=2 > W-timestamp(A)=0. $\therefore$ W-timestamp(A) $\leftarrow$ 2
- T2: R(B) $\rightarrow$ OK because TS(T2)=2 > W-timestamp(B)=0. $\therefore$ R-timestamp(B) $\leftarrow$ 2
- T1: W(B) $\rightarrow$ Rejected because TS(T1)=1 < R-timestamp(B)=2.

(b) If T2 does not read B, i.e., if R(B) is not called by T2, can we transform the schedule into a serial schedule? If so, why?

- T1: R(A) $\rightarrow$ OK because TS(T1)=1 > W-timestamp(A)=0 and TS(T1)=1 > R-timestamp(A)=0. $\therefore$ R-timestamp(A) $\leftarrow$ 1
- T2: R(A) $\rightarrow$ OK because TS(T2)=2 > W-timestamp(A)=0 and TS(T1)=1 > R-timestamp(A)=1. $\therefore$ R-timestamp(A) $\leftarrow$ 2
- T2: W(A) $\rightarrow$ OK because TS(T2)=2 > W-timestamp(A)=0. $\therefore$ W-timestamp(A) $\leftarrow$ 2
- T1: W(B) $\rightarrow$ OK because TS(T1)=1 > R-timestamp(B)=0 and TS(T1)=1 > W-timestamp(B)=0. $\therefore$ W-timestamp(B) $\leftarrow$ 1
- T2: W(B) $\rightarrow$ OK because TS(T2)=2 > R-timestamp(B)=0 and TS(T2)=2 > W-timestamp(B)=1. $\therefore$ W-timestamp(B) $\leftarrow$ 2

So, this is a valid schedule, and we can execute all operations of T1 first, and then all operations of T2 later because there' no conflicting operations. Hence, this schedule is serializable.

13. [Multi-version Timestamp Ordering] Is the following schedule valid under multi-version timestamp ordering protocol? Justify your answer.

T1	T2
R(A)	
A=A+1	
	R(A)
W(A)	
	A=A+1
	W(A)

Figure 5: Multi-version timestamp ordering protocol

No, it is not. T1's W(A) will find out that R-timestamp(A)=2 since it is read by T2. ∴ T1's W(A) will be rejected and T1 should abort under the multi-version timestamp ordering protocol.

14. [Recovery] Suppose the system crashed after Log Sequence No 9. The checkpoint file created by Log Sequence No 8 is valid.

```
LogSeqNo Log
  1      <T1 Start>
  2      <T2 Start>
  4      <T1, X, 0, 100>
  5      <T1, Z, 0, 101>
  6      <T1 commit>
  7      <T2, X, 100, 10>
  8      <checkpoint {T2}>
  9      <T2, Y, 0, 20>
 >>>     CRASH!!!!
```

(a) What log entries will the recovery scheme append to recover from the failure?

<T2, X, 100>¡T2, Y, 0¿ ¡T2, abort¿

(b) How many times X, Y, and Z will be written by the undo-redo recovery process? Justify your answer.

x → 100 // undo y → 20 //redo → 0 // undo
X: once Y: Twice

15. [Big Data] Give short answers for the following questions.

(a) Why are LSM-trees faster than B+trees for insertions?

upper(lower) level indexing components are small.

(b) Why is Hadoop so popular even though it only supports two simple operations?

scalability

developers do not need to worry about complicated details of distributed syste

(c) Under what conditions, will you use MongoDB instead of MySQL?

development process is flexible. (no fixed schema is required.)

scalability

Simple queries

Functionality provided applicable to most web applications

Easy and fast integration of data

No ERD diagram

Not well suited for heavy and complex transactions systems