Carnegie Mellon University
Department of Computer Science
15-415/615- Database Applications
C. Faloutsos & A. Pavlo, Spring 2015
Prepared by Hong Bin Shim
DUE DATE: Thu, 4/23/2015, 1:30pm

Homework 8

IMPORTANT

- **Deposit hard copy** of your answers in **class at 1:30pm on Thu, 4/23/2015**.
- Separate answers, as usually, i.e., please each question on a separate page, with the usual info (andrewID, etc)

Reminders

- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **Typeset** all of your answers whenever possible. Illegible handwriting may get no points, at the discretion of the graders.
- **Late homeworks**: please email late homeworks
  - to all TAs
  - with the subject line exactly `15-415 Homework Submission (HW 8)`
  - and the count of slip-days you are using.

For your information:

- Graded out of **100** points; **4** questions total
- Rough time estimate: ≈4 hours (∼1 hour for each question)

Revision : 2015/04/14 08:13

Question	Points	Score
Serializability and 2PL	20	
Deadlock Detection and Prevention	30	
Hierarchical Locking - Oscar Nominations	30	
B+ tree Locking	20	
Total:	100	

Question 1: Serializability and 2PL . . . . . [20 points]

Submit on separate page
Course: 15-415/615; HW: ; Q:
Name: _____; andrew-id: _____; late days:

(a) Yes/No questions:

i. **[2 points]** Conflict Serializability always permits as many schedules as View Serializability does.
☐ Yes ☐ No

ii. **[2 points]** A schedule is conflict serializable if its dependency graph is acyclic.
☐ Yes ☐ No

iii. **[2 points]** In the Shrinking phase of 2PL, the transaction can be granted lock requests.
☐ Yes ☐ No

iv. **[2 points]** All serializable schedules are allowed by 2PL.
☐ Yes ☐ No

v. **[2 points]** Schedules under strict 2PL could have cascading aborts.
☐ Yes ☐ No

(b) Serializability:
Consider the schedule given below in Table 1. R(·) and W(·) stand for 'Read' and 'Write', respectively.

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
T_1			R(A)		W(A)		R(C)		W(C)		
T_2				R(B)		W(B)					
T_3	R(A)	W(A)						R(B)	W(B)	R(C)	W(C)

Table 1: A schedule with 3 transactions

i. **[1 point]** Is this schedule serial?
☐ Yes ☐ No

ii. **[3 points]** Give the dependency graph of this schedule.

iii. **[1 point]** Is this schedule conflict serializable?
☐ Yes ☐ No

iv. **[3 points]** If you answer "yes" to (iii), provide the equivalent serial schedule. If you answer "no", briefly explain why.

v. **[2 points]** Could this schedule have been produced by 2PL?
☐ Yes ☐ No

Question 2: Deadlock Detection and Prevention ..... [30 points]

Submit on separate page
Course: 15-415/615; HW: ; Q:
Name: _____; andrew-id: _____; late days:

(a) Deadlock Detection:
Consider the following lock requests in Table 2. And note that

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, and T_3 represent three transactions.
- LM stands for 'lock manager'.

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7
T_1	S(D)	S(A)			X(C)		S(B)
T_2			S(A)	X(D)			
T_3						S(C)	
LM	g						

Table 2: Lock requests of 3 transactions

i. **[6 points]** For the lock requests in Table 2, determine which lock will be granted or blocked by the lock manager. Please write 'g' in the LM row to indicate the lock is granted and 'b' to indicate the lock is blocked. For example, in the table, the first lock (S(D) at time t_1) is marked as granted.

ii. **[4 points]** Give the wait-for graph for the lock requests in Table 2.

iii. **[3 points]** Determine whether there exists a deadlock in the lock requests in Table 2, and briefly explain why.

(b) Deadlock Prevention:
Consider the following lock requests in Table 3. Again,

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, T_3, and T_4 represent four transactions.
- LM represents a 'lock manager'.

time	t_1	t_2	t_3	t_4	t_5	t_6
T_1	S(D)		S(B)			
T_2				S(C)		X(D)
T_3		X(B)				
T_4					X(C)	
LM	g					

Table 3: Lock requests of 4 transactions

i. **[6 points]** For the lock requests in Table 3, determine which lock request will be granted, blocked or aborted by the lock manager (LM), if it has no deadlock prevention policy. Please write 'g' for grant, 'b' for block and 'a' for abort. Again, example is given in the first column.

ii. **[5 points]** Give the wait-for graph for the lock requests in Table 3. Determine whether there exists a deadlock in the lock requests in Table 3 under LM, and briefly explain why.

iii. **[3 points]** To prevent deadlock, we use the lock manager (LM) that adopts the Wait-Die policy. We assume that in terms of priority: $T_1 > T_2 > T_3 > T_4$. Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a'). Follow the same format as the previous question.

iv. **[3 points]** Now we use the lock manager (LM) that adopts the Wound-Wait policy. We assume that in terms of priority: $T_1 > T_2 > T_3 > T_4$. Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a'). Follow the same format as the previous question.

Question 3: Hierarchical Locking - Oscar Nominations[30 points]

Submit on separate page
Course: 15-415/615; HW: ; Q:
Name: _____; andrew-id: _____; late days:

Consider a Database (D) consisting of two tables, Actors (A) and Nominations (N). Specifically,

- Actors(<u>aid</u>, first_name, last_name), spans 300 pages, namely A_1 to A_{300}
- Nominations(<u>nid</u>, aid, year, movie, character, won), spans 600 pages, namely N_1 to N_{600}

Further, **each page contains 100 records**, and we use the notation $A_3 : 20$ to represent the 20^{th} record on the third page of the Actors table. Similarly, $N_5 : 10$ represents the 10^{th} record on the fifth page of the Nominations table.
We use Multiple-granularity locking, with **S, X, IS, IX** and **SIX** locks, and **four levels of granularity**: (1) *database-level (D)*, (2) *table-level (A, N)*, (3) *page-level ($A_1 - A_{300}$, $N_1 - N_{600}$)*, (4) *record-level ($A_1 : 1 - A_{300} : 100$, $N_1 : 1 - N_{600} : 100$)*.
For each of the following operations on the database, please determine the sequence of lock requests that should be generated by a transaction that want to carry out these operations efficiently.
Please follow the format of the examples listed bellow:

- write **"IS(D)"** for a request of **database-level IS lock**
- write **"X($N_2 : 30$)"** for a request of **record-level X lock for the** 30^{th} **record on the second page of the Nominations table**
- write **"S($N_2 : 30 - N_3 : 100$)"** for a request of **record-level S lock from the** 30^{th} **record on the second page of the Nominations table to the** 100^{th} **record on the third page of the Nominations table**.

(a) **[6 points]** Read 100^{th} record on page A_1.

(b) **[6 points]** Read ALL records on page A_1 through A_{15}, and modify the records $A_{10} : 10$ through $A_{10} : 100$.

(c) **[6 points]** Modify the first record on EACH and EVERY page of the Nominations table (these are blind writes that do not depend on the original contents in the pages).

(d) **[6 points]** For EACH record in the Actors table, capitalize the English letters in the **last_name** if it is not capitalized. That is, "Redmayne" will be modified as "REDMAYNE" but "MOORE" will be left unchanged.

(e) **[6 points]** Delete ALL the records from ALL tables.

Question 4: B+ tree Locking . . . . . [20 points]

Submit on separate page
Course: 15-415/615; HW: ; Q:
Name: _____; andrew-id: _____; late days:

Consider the following B+ tree:

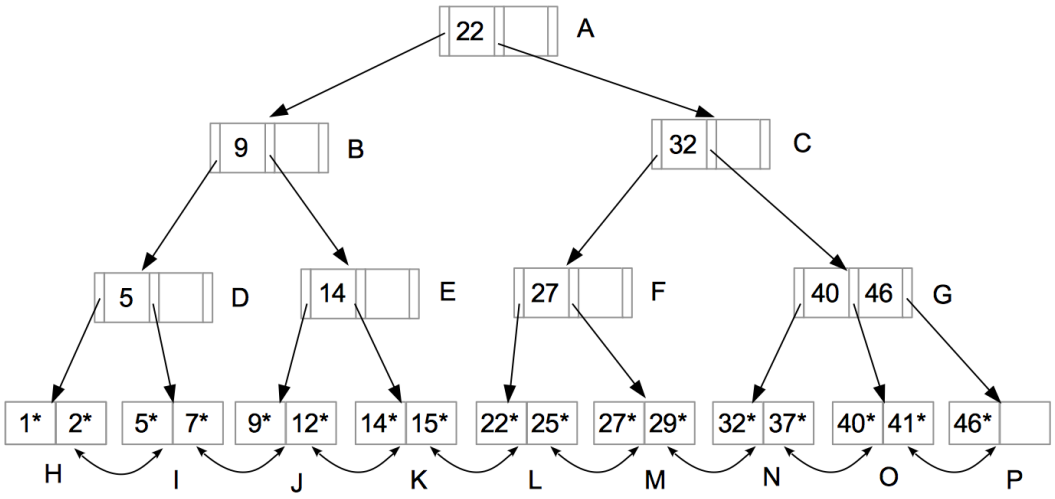

Figure 1: B+ tree locking

To lock this B+ tree, we would like to use the **Bayer-Schkolnick** algorithm (described in lecture notes #22[1], slide 32 - 35). **Important**: we use the version as presented in the lecture, which **does not** use lock upgrade.
For each of the following transactions, give the sequence of lock/unlock requests. For example, please write $S(A)$ for a request of shared lock on node A, $X(B)$ for a request of exclusive lock on node B and $U(C)$ for a request of unlock node C.
Important notes:

- Each of the following transactions is applied on the *original tree*, i.e., please ignore any change to the tree from earlier problems.
- For simplicity, *ignore* the changes on the pointers between leaves.

(a) **[5 points]** Search for data entry "37*"

(b) **[5 points]** Delete data entry "7*"

(c) **[5 points]** Insert data entry "30*"

(d) **[5 points]** Insert data entry "47*"

[1] `http://www.cs.cmu.edu/~christos/courses/dbms.S15/slides/22CC2.pdf`