Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos & A. Pavlo, Fall 2016

Homework 8 (by Prashanth Menon) - Solutions
Due: **hard copy, in class** at **3:00pm, on Monday, Dec. 5**

VERY IMPORTANT: Deposit **hard copy** of your answers, in class. For ease of grading, please

1. **Separate** your answers, on different page(s) for each question (staple additional pages, if needed).
2. **Type** the full info on **each** page: your **name**, **Andrew ID**, **course#**, **Homework#**, **Question#** on each of the 3 pages.

Reminders:

- *Plagiarism*: Homework is to be completed *individually*.
- *Typeset* all of your answers whenever possible. Illegible handwriting may get zero points, at the discretion of the graders.
- *Late homeworks*: in that case, please email it
  - 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; **3** questions total
- Rough time estimate: *approx. 6 hours* - 1 to 2 hours per question

Revision : 2016/12/10 16:05

Question	Points	Score
Serializability and 2PL	33	
Deadlock Detection and Prevention	34	
Hierarchical Locking - Return of Bike Sharing	33	
Total:	100	

Question 1: Serializability and 2PL .... [33 points]

GRADED BY:

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

(a) Yes/No questions:

i. [**3 points**] Every conflict-serializable schedule is view-serializable.
■ **Yes** ☐ No

ii. [**3 points**] In the shrinking phase of strict 2PL, locks cannot be released until the end of the transaction.
■ **Yes** ☐ No

iii. [**3 points**] Schedules under strict 2PL do not allow dirty reads.
■ **Yes** ☐ No

iv. [**3 points**] Schedules under strict 2PL may lead to cascading aborts.
☐ Yes ■ **No**

v. [**3 points**] Only schedules under 2PL (and not strict 2PL) may lead to deadlocks.
☐ Yes ■ **No**

Grading info: -3 for each incorrect answer

(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(B)		W(B)
T_2				R(C)	R(A)		W(A)			W(C)	
T_3		R(B)				W(B)		R(A)			

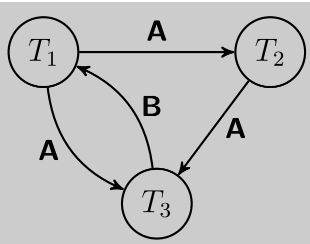
Table 1: A schedule with three transactions: T_1, T_2, and T_3

i. [**2 points**] Is this schedule serial?
☐ Yes ■ **No**

Grading info: -2 for incorrect answer

ii. [**5 points**] Give the dependency graph of this schedule.

Solution:



Grading info: No partial credit awarded unless A, B marked, -1 for each missing/incorrect edge otherwise, -1 for mislabeled nodes in an otherwise correct graph.

iii. [**2 points**] Is this schedule conflict serializable?
☐ Yes ■ **No**
Grading info: -2 for incorrect answer

iv. [**2 points**] Is this schedule view serializable?
☐ Yes ■ **No**
Grading info: -2 for incorrect answer

v. [**5 points**] If you answer "yes" to (iii), provide the equivalent serial schedule. If you answer "no", briefly explain why.

Solution: The schedule is not serializable because there are two cycles in the dependency graph: $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ **and** $T_1 \rightarrow T_3 \rightarrow T_1$.

Grading info: -3 for not stating that the schedule is not serializable.
-2 for not providing correct justification or not listing both cycles.

vi. [**2 points**] Could this schedule have been produced by 2PL?
☐ Yes ■ **No**
Grading info: -1 for incorrect answer

Question 2: Deadlock Detection and Prevention . . . . . [34 points]

GRADED BY:

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

(a) Deadlock Detection:
Consider the following lock requests in Table 2. 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	X(A)						S(C)
T_2			S(B)	S(C)		S(A)	
T_3		S(C)			X(B)		
LM	g						

Table 2: Lock requests of three transactions: T_1, T_2, and T_3

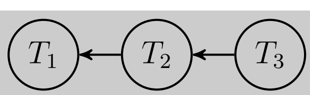
i. [**6 points**] For the lock requests in Table 2, determine which lock will be granted or blocked by the lock manager. Please write 'g' in the LM row to indicate the lock is granted and 'b' to indicate the lock is blocked. For example, in the table, the first lock (X(A) at time t_1) is marked as granted.

Solution:

- S(C) at t_2: g
- S(B) at t_3: g
- S(C) at t_4: g
- X(B) at t_5: b
- S(A) at t_6: b
- S(C) at t_7: g

Grading info: Half points for one mistake in the schedule, no points > 1 mistake.

ii. [**5 points**] Give the wait-for graph for the lock requests in Table 2 at time-tick t_7.

Solution: 

Grading info: Half points for 1 missing directed edge, no points if missing > 1.

iii. [**4 points**] Determine whether there exists a deadlock in the lock requests in Table 2, and explain why.

Solution: A dead lock does not exists because there is no cycle in the dependency graph.

Grading info: -3 *points for not explaining why there is no deadlock*

(b) Deadlock Prevention:
Consider the following lock requests in Table 3. As before:

- 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_7	t_8
T_1	X(B)			S(A)				
T_2					X(D)	X(C)		
T_3			S(C)				X(B)	
T_4		X(A)						S(D)
LM	g							

Table 3: Lock requests of four transactions: T_1, T_2, T_3, and T_4

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*; for 'abort', specify which tranaction is aborted - e.g., 'a' (T1 is aborted) An example is given in for time-tick t_1.

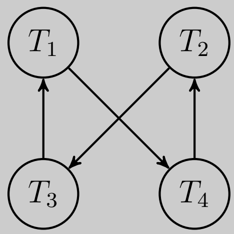
Solution:

- X(A) at t_2: g
- S(C) at t_3: g
- S(A) at t_4: b
- X(D) at t_5: g
- X(C) at t_6: b
- X(B) at t_7: b
- S(D) at t_8: b

Grading info: Half points for one mistake in the schedule, no points > 1 *mistake.*

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 explain why.

Solution: A deadlock exists because there is a cycle in the dependency graph.



Grading info: *−3 points for not stating that a deadlock exists*
−2 points for not explaining why there is a deadlock, or not showing a cycle

iii. [**4 points**] To prevent deadlock, we use a lock manager (LM) that adopts the Wait-Die policy. We assume the four transactions have priority: $T_1 < T_2 < T_3 < T_4$. *Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a')*; for 'abort', specify which tranaction is aborted - e.g., 'a' (T1 is aborted). Follow the same format as the previous question.

Solution:

- X(A) at t_2: g
- S(C) at t_3: g
- S(A) at t_4: a (T_1 aborts.)
- X(D) at t_5: g
- X(C) at t_6: a (T_2 aborts.)
- X(B) at t_7: g (No locks held on B since T_1 was aborted.)
- S(D) at t_8: g (No locks held on D since T_2 was aborted.)

Grading info: *−2 points for one mistake in the schedule, no points > 1 mistake.*

iv. [**4 points**] In this question, we use a lock manager (LM) that adopts the Wound-Wait policy. We assume the four transactions have priority: $T_1 < T_2 < T_3 < T_4$. *Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a')*; for 'abort', specify which tranaction is aborted - e.g., 'a' (T1 is aborted) Follow the same format as the previous question.

Solution:

- X(A) at t_2: g

- S(C) at t_3: g
- S(A) at t_4: b
- X(D) at t_5: g
- X(C) at t_6: b
- X(B) at t_7: g (T_1 aborts.)
- S(D) at t_8: g (T_2 aborts.)

Grading info: -2 *points for one mistake in the schedule, no points* > 1 *mistake.*

Question 3: Hierarchical Locking - Return of Bike Sharing[33 points]

GRADED BY:

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

For this problem we consider a modified and simplified version of the bike sharing database from Homework 2. The bike sharing database has the following three tables:

Our bike sharing database (D) contains three tables: Bike (B), Station (S), and Trips (T). Specifically:

- `Bikes(bid, model, year)`, that spans 150 pages, namely B_1 to B_{150}.
- `Trips(tid, date, start_city, end_city, distance, bid)`, that spans 600 pages, namely T_1 to T_{600}.

Each page contains 100 records, and we use the notation $B_i : j$ to represent the j^{th} record, $1 \leq j \leq 100$, on the i^{th} page of table B. For example, $B_5 : 10$ represents the tenth record on the fifth page of the `Bikes` 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 (B, S, T)*, (3) *page-level* ($B_1 - B_{150}, T_1 - T_{600}$), (4) *record-level* ($B_1 : 1 - B_{150} : 100, T_1 : 1 - T_{600} : 100$).

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. You do not need to list unlock requests.

Please follow the format of the examples listed below:

- Write **"IS(D)"** to request a **database-level IS lock**
- Write **"X(B_2 : 30)"** to request a **record-level X-lock for the 30^{th} record on the second page of the `Bikes` table**
- Write **"S($T_2 : 30 - T_3 : 100$)"** to request a **record-level S-lock from the 30^{th} record of the second page of the `Trips` table to the 100^{th} record of the third page of the `Trips` table**.

(a) [**7 points**] Calculate the average distance of all trips.

> **Solution:** IS(D), S(T)
>
> *Grading info:* -3 *for each missing/incorrect mistake*

(b) [**6 points**] Read ALL records on page B_{10} through B_{70}, and modify the record $B_{11} : 44$.

> **Solution:** IX(D), SIX(B), IX(B_{11}), X($B_{11} : 44$);
> also acceptable: IX(D), IX(B), S(B_{10}), S($B_{12} - B_{70}$), SIX(B_{11}), X($B_{11} : 44$)
>
> *Grading info:* -2 *for each missing/incorrect mistake (upto 6 points)*

Question 3 continues. . .

(c) [**7 points**] Modify the `date` attribute of the last record on EACH and EVERY page of the `Trips` table to today's date.

Solution: IX(D), X(T);
also acceptable: IX(D), IX(T), IX($T_1 - T_{600}$), X($T_1 : 100 - T_{600} : 100$)
Grading info: -2 *for each missing/incorrect mistake (upto 6 points)*

(d) [**7 points**] Incremement the `distance` attribute of all records from the `Trips` table whose `start_city` is 'Pittsburgh'.

Solution: IX(D), X(T);
also acceptable: IX(D), SIX(T), then request record-level X locks on any records that need to be updated.
Grading info: -3 *for each missing/incorrect mistake*

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

Solution: X(D)
Grading info: -6 *for each missing/incorrect mistake*