UCLA
Computer Science Department
Fall 2003

Instructor: C. Zaniolo
TA: Fusheng Wang

Student Name and ID: ____________________

CS143 Final EXAM: Closed Book, 3 Hours

Attach extra pages as needed. Write your name and ID on any extra page that you attach. Please, write neatly.

Problem	**Score**	
1	(24%)	
2	(25%)	
3	(30%)	
4	(21%)	
Total	(100%)	

Extra Credit:

Final Score:

Problem 1. 24 Points

In a distributed DB, there are three sites, A, B, C, and there are communication lines between A and B and between B and C, but no direct line between A and C. The three sites co-operate in a distributed transaction using the Two-Phase Commit (2PC) protocol; A is the coordinator.

If you agree with the following statements, just write 'TRUE'. If you disagree, write 'FALSE' and explain the reasons why the statement is false.

The communication lines: A — B — C

A. If the communication is lost between A and B, A waits indefinitely in a blocked state till communication is restored.

   FALSE: A is the coordinator. If the communication stops before it receives the 'vote to commit' by all the sites, the coordinator waits for a while, but then it aborts and instructs the participants to do the same. Once the coordinator receives commit from all participants the coordinator just commit and sent them back the commit order.

B. Say that B receives the `commit` instruction from A but cannot forward such instruction to C since the line between B and C is down. Then B must wait in a blocked state till the line is restored and C can be sent the instruction to commit.

   FALSE: B commits. In general, the sites act on the basis of the messages they receive, not on the basis of the messages they can send (since they can never be sure on the state of lines).

Now say that B receives the `commit` instruction from A, but immediately after that, both lines between A and B and between B and C go down. Say that the line between B and C is restored well before the line between B and A. Then:

C. B remains in a blocked state till the line with A is restored, and then asks A for instructions.

   FALSE: B commits. Same reason as in the previous case B.

D. C remains in a blocked state till its line with B is restored, and then it ask B for its decision and replicates it.

   TRUE. Since C never received the commit message, it will have to ask B what happened and follow suite.

Problem 2, 25 points: Consider the following schedule (where T0 is older than T1, which is older than T2).

```
    T0            T1            T2
 --------------------------------------
   write(A)
                read(B)
   read(B)
                             write(B)
                read(A)
                write(A)
                            request a lock
                            on A and abort
   read(C)
                write(C)
   read(D)
 ---------------------------------------
```

a) Draw the precedence graph and list all the equivalent serial schedules for the completed transactions.
Precedence Graph: `T0--A,C--> T1`; T2 does not complete, it aborts.
The only equivalent serial schedule is T0 followed by T1.

b) Which of the following policies could have generated this schedule?

(1) A 2PL protocol? (show a complete schedule with the lock-X, lock-S and unlock for each resource)
Assume 2PL. At the point where T2 performs `write(T2)` T0 and T1 must have released the locks they previously had on `B`. Thus, at this point, `T0` and `T1` are in the shrinking phase where no new lock can be obtained. Since, at this point, only one has has a lock on `C` and no new lock can obtained later, they cannot both perform the operations shown in the schedule. Thus, this is not a 2PL protocol.

(2) a strict 2PL protocol? (show a complete schedule with the lock-X, lock-S and unlock for each resource)
N0: This is not 2PL, and is not strict. The schedule show that T1 locks A before T0 has completed; but in a strict 2PL T0 would have kept the lock till the end of the transaction.

(3) 2PL with wait-die deadlock prevention strategy?
NO: T0 and T1 do not follow 2PL. However, T2 follows the wait-die strategy: the younger transaction T2 finds B locked by the older T1 and thus abort and restarts later with its old timestamp.

(4) 2PL with a wound-wait deadlock prevention strategy?
NO: T0 and T1 do not follow 2PL, and no with wound-wait the younger T2 would have waited until A is unlocked by the older T1.

(5) Cascading of rollbacks to kill transactions that performed ‘dirty reads’
NO. A ’dirty read’ is on piece of data changed by a different transaction that has then aborted. Here T2 aborts, but nobody reads the data written by T2.

Problem 3, 30 points: We have a relation schema $R(C, S, J, D, P, Q, V)$ with the following FDs:

$C \rightarrow CSJDPQV$
$JP \rightarrow C$
$SD \rightarrow P$
$J \rightarrow S$

a) Find a canonical cover for these FDs (show the steps)

1. $JP \rightarrow C$
2. $SD \rightarrow P$
3. $J \rightarrow S$
4. $C \rightarrow S$
5. $C \rightarrow J$
6. $C \rightarrow D$
7. $C \rightarrow P$
8. $C \rightarrow Q$
9. $C \rightarrow V$

No extraneous attribute in the left side of the first two FDs. However, FDs 4. and 7. are redundant.

b) Use the 3NF design algorithm to design a 3NF schema that has the lossless-join property and a minimal relation count (show the steps)

$\mathrm{R_1}(\underline{\mathrm{J, P}}, \mathrm{C})$; $\mathrm{R_2}(\underline{\mathrm{S, D}}, \mathrm{P})$; $\mathrm{R_3}(\underline{\mathrm{J}}, \mathrm{S})$; $\mathrm{R_4}(\underline{\mathrm{C}}, \mathrm{J})$; $\mathrm{R_5}(\underline{\mathrm{C}}, \mathrm{D})$; $\mathrm{R_6}(\underline{\mathrm{C}}, \mathrm{Q})$; $\mathrm{R_7}(\underline{\mathrm{C}}, \mathrm{V})$.

Lossless join property: C is a key for the original relation and the 3NF decomposition. No additional relation needed to ensure the lossless join property.

Minimal relation count: all the relations with candidate key C can be merged.
Also $C \leftrightarrow JP$: thus the first relation can be joined with those having C as Key. We obtain:

$\mathrm{R_{14567}}(\underline{\mathrm{J, P}}, \underline{\underline{\mathrm{C}}}, \mathrm{D, Q, V})$; $\mathrm{R_2}(\underline{\mathrm{S, D}}, \mathrm{P})$; $\mathrm{R_3}(\underline{\mathrm{J}}, \mathrm{S})$.

c) Is the resulting schema BCNF (explain your answer)?

Yes. FD 3 is no longer contained in any relations. All the remaining FDs are in relations where their left side is a superkey (an actual key for all the FDs at hand).

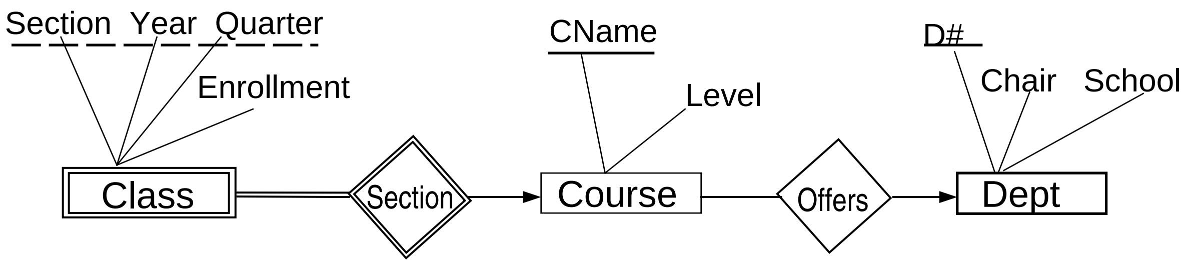
Problem 4, 21 points:

`Course(CName, Level, D#),`

`Dept(D#, Chair, School),`

`Class(CName, Section, Year, Quarter, Enrollment)`

a) Draw the E-R diagram from which this 3NF schema could have been produced.



b) Now assume that currently there are 20 schools, 100 departments, 4000 different courses, and two sections for each course. Every school offers some courses.

Indicate what primary and secondary indices should be created to support the following two queries efficiently:

Q1: Find the chair of the department offering a given course,

Index `Course.Cname` to find `D#` for the given course. Then to find the chair of the department `D#` just found we can use an index `Dept.D#`. Each search finds one record, at most. There is no benefit in clustering the file by the key of an index supporting exact-match searches for a single record.

Q2: Finds all courses offered in a given school.

To support the search for the given `School`, we should have an index on `Dept.School`; then, using `D#` so found we should have. index on `Course.D#` to expedite the join. On the average, there are five departments per each school, and 40 courses per department. Thus, both searches are greatly expedited by clustering these two indexes—i.e., we should make both of them primary indexes.

c) Estimate the average number of tuples produced by Q2.

Say that D# is a key foreign key. Then the natural join of Course and School contains 4000 different tuples. Also there are 20 different schools in the join. Then Q2 returns 4000/20=200.

Otherwise, we can assume 5 departments per school and 40 courses per department. Then Q2 returns $5 \times 40 = 200$ tuples.

Extra Credit Problem, 4 points: On the average, should you expect fewer rollbacks with the wait-die protocol or the wound-wait protocol? (Justify your answer)

In the wait-die, when the younger transaction dies and is restarted immediately it is likely to find the same resource still occupied. Then it will die again, and this situation could occur several times. For wound-wait, the younger transaction (Y) can be killed by an older transaction (O). When restarted, Y is likely to find the resource still occupied by the its killer O: but now, Y is the younger of the two and waits, and it gets the resource as soon as O is done.