Transactions

R(A, B)

A	B
1 2	10 20

Consider the following schedule with *SERIALIZABLE* isolation level:

	T1	T2	T3
1	begin transaction;		
2	select B from R;		
3		begin transaction;	
4		select * from R where A = 2;	
5	update R set B = 30 where A = 2;		
6		select * from R where A = 2;	
7	commit;		
8			begin transaction;
9			select * from R where A = 2;
10		commit;	
11			
12			
13			commit;

Indicate the status for each the command, if it is a SUCCESS, ERROR or WAIT. Also indicate the values of B returned from the command, if it is a SUCCESS. If the command has an ERROR, indicate when should the command continue. Execute the transactions based on the following DBMS: SQLite and SQL Azure

Answer:

SQLite			
	T1	T2	T3
1	begin transaction; *SUCCESS*		
2	select * from R; *readlock* *SUCCESS, values: 10, 20*		
3		begin transaction; *SUCCESS*	
4		select * from R where A = 2; *readlock* *SUCCESS, values: 10, 20*	
5	update R set B = 30		

	where A = 2; - writelock *SUCCESS*		
6		select * from R where A = 2; *SUCCESS, values: 20*	
7	commit;		
8			begin transaction;
9			select * from R where A = 2; *ERROR: cannot get readlock because T1 is holding a writelock*
10		commit; *SUCCESS*	
11	Try commit again: commit; *SUCCESS*		
12			select * from R where A = 2; *SUCCESS, value 30*
13			commit; *SUCCESS*

SQL Azure			
	T1	T2	T3
1	begin transaction; *SUCCESS*		
2	select * from R; *readlock* *SUCCESS, values: 10, 20*		
3		begin transaction; *SUCCESS*	
4		select * from R where A = 2; *readlock* *SUCCESS, values: 10, 20*	
5	update R set B = 30 where A = 2; - writelock *SUCCESS*		
6		select * from R where A = 2; *wait for T1 to commit*	
7	commit; SUCCESS	*SUCCESS, values: 30*	

8			begin transaction;
9			select * from R where A = 2; *SUCCESS, values: 30*
10		commit; *SUCCESS*	
11			
12			
13			commit; *SUCCESS*

Consider the following schedules, indicate if it is *conflict-serializable* or not (Fall 2012)

i) r1(A); r2(B); r1(B); w2(B); w1(A); w1(B); r2(A); w2(A); c1; c2 ***NO***
ii) r1(A); r2(B); r3(A); r2(A); r3(C); r1(B); r3(B); r1(C); r2(C); c1; c2; c3 ***YES***
iii) w1(A); w2(A); w1(B); w3(B); w1(C); w3(C); w2(C); c1; c2; c3 ***YES***

Consider the following java pseudocode, explain the problem with this approach and how to fix it

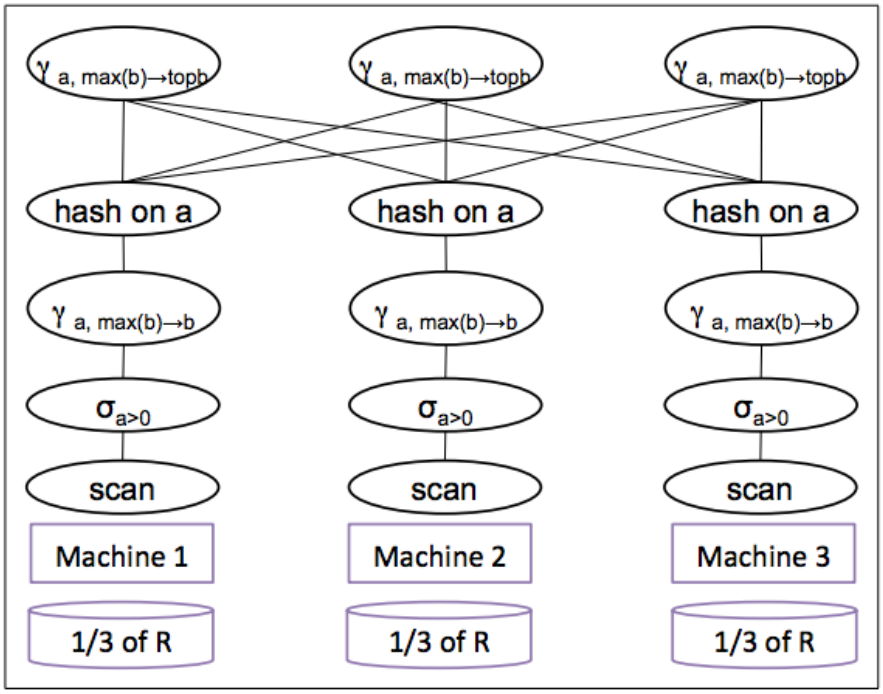
```
i)   Prompt the user for a student ID and password.

ii)  Start a new transaction.

iii) Look up the student in the database.

iv)  If the student ID is not in the database or the
     password is incorrect, abort the transaction.

v)   Look up the courses recommended for the student.
     Display the courses on the screen.

vi)  While the user does not choose QUIT

     a. Prompt the user to select a course.

     b. If the course is available, then register the
        student.

vii) End while

     Commit the transaction.
```

Answer: The transaction is too coarse grained. A lock will be hold for a very long time, because of the while-loop. To fix it, modify the transaction to be more fine-grained.

Parallel Data Processing
Given the following query, show a relational algebra plan for this query. There are 3 machines and the data is evenly spread across each machine.

```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a;
```

Answer:



If we modify the query to the following query, will the plan change?

```
SELECT a, avg(b) as avgb
FROM R
WHERE a > 0
GROUP BY a;
```

Answer: Yes, we cannot do the local aggregation before hashing to spread the data.

Describe how the first query will be executed using Map-Reduce (Describe the map-reduce function)

Answer: map() apply the predicate on a and emit; a: [list of all possible Bs]
reduce() iterate through all values of B is find the max value.