CS 186/286 Spring 2018 Midterm 2

- Do not turn this page until instructed to start the exam.
- You should receive 1 single-page *answer sheet* and a 15-page *exam packet*.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- You have *80 minutes* to complete the midterm.
- The midterm has *4 questions*, each with multiple parts.
- The midterm is worth a total of *60 points*.
- For each question, place only your *final answer* on the answer sheet; do not show work.
- For multiple choice questions, please fill in the bubble or box completely, **do not mark the box with an X or checkmark**.
- Use the blank spaces in your exam for scratch paper.
- You are allowed **one** 8.5" $\times$ 11" double-sided page of notes.
- No electronic devices are allowed.

1 Joins (20 points)

Note: Assume that when we write $X \bowtie Y$, X (on the left) is the outer, and Y (on the right) is the inner for the remainder of these questions.

1. (1 point) Suppose we do a Page-oriented Nested Loop Join on two tables, P and Q. We are told that the same amount of pages are required to store P and Q, but Q has three times more records per page than P, on average. Which one of the following options is true:

   A. $P \bowtie Q$ does fewer I/Os than $Q \bowtie P$.

   B. $Q \bowtie P$ does fewer I/Os than $P \bowtie Q$.

   C. $P \bowtie Q$ does the same number of I/Os as $Q \bowtie P$.

   Solution: What is the cost for PNLJ on $P \bowtie Q$? $[P] * [Q] + [P]$.
   What is the cost for PNLJ on $Q \bowtie P$? $[Q] * [P] + [Q]$.
   But we are told that $[P] == [Q]$.
   Therefore C.

2. (5 points) Next, we consider a Simple Nested Loop Join on three tables: P, Q, and S.

   - Please use $[X]$ to denote the number of pages of X, and $\{X\}$ to denote the number of records in X (note the curly braces rather than vertical bars, for legibility).
   - Do not materialize intermediary results into a temporary table. Assume the results of a subplan are pipelined to the next operator.
   - Assume $P \bowtie Q$ is foreign key to primary key join, and the following join with S is also a foriegn key to primary key join. The (non null) foreign keys are in P, and primary keys are in Q and S.

   (a) (1 point) What is the I/O cost for $P \bowtie Q$?

   Solution: $\{P\} * [Q] + [P]$

   (b) (1 point) What is the cardinality of $P \bowtie Q$?

   Solution: Every record in P has a single matching record in Q because of the foreign key to primary key constraint. Thus, the result size is simply $\{P\}$.

   (c) (2 points) What is the I/O cost for $(P \bowtie Q) \bowtie S$?

   Solution: $\{P\} * ([Q] + [S]) + [P]$
   This is the cost of $P \bowtie Q$ plus $\{P \bowtie Q\} * [S]$.
   Using the result from above, this implies $\{P\} * [Q] + [P]$ plus $\{P\} * [S]$.
   Therefore, $\{P\} * ([Q] + [S]) + [P]$.

   (d) (1 point) What is the cardinality of $(P \bowtie Q) \bowtie S$?

   Solution: Every record in $(P \bowtie Q)$ has a single matching record in S because of the foreign key to primary key constraint between P and Q. Thus, the result size is $\{P\}$.

3. (1 point) Which one of the following options is true about the IO cost of a self join (i.e. joining a table with itself)? Assume $B \gg 3$ where B is the pages of memory available.

   A. IO(SNLJ) > IO(PNLJ) ≥ IO(BNLJ)

   B. IO(SNLJ) ≤ IO(PNLJ) ≤ IO(BNLJ)

   C. IO(SNLJ) ≥ IO(PNLJ) ≥ IO(BNLJ)

   D. IO(PNLJ) ≤ IO(SNLJ) ≤ IO(BNLJ)

> **Solution:** C. The I/O cost of BNLJ is guaranteed to be at least as good as PNLJ and SNLJ. PNLJ is guaranteed to be at least as good as SNLJ.

4. (3 points) Suppose there are two tables: R with 40 pages and S with 75 pages. What is the cost of sort-merge join between tables R and S? Assume the operator has 6 pages of RAM, and it is not writing the output to disk. Use the optimization from lecture if possible.

> **Solution:** After the first sorting pass —
>
> There are 7 runs of R, 6 of 6 pages and 1 of 4 pages.
>
> There are 13 runs of S, 12 of 6 pages and 1 of 3 pages.
>
> After the second sorting pass –
>
> There are 2 runs of R, 1 of 30 pages and 1 of 10 pages.
>
> There are 3 runs of S, 2 of 30 pages and 1 of 15 pages.
>
> Then —
>
> We can merge R and S in the final pass because there are 5 total runs and 5 input buffers.
>
> $$2 * 2 * ([R] + [S]) + ([R] + [S]) = 4 * 115 + 115 = 575$$

5. (10 points) Suppose there are two tables: P and Q. Table P has 1,200 pages and 35 records per page. Table Q has 100 pages and 50 records per page. In this question, we consider an equality join $P \bowtie Q$. Assume that join key values are widely and uniformly distributed. That is, there is no skew. Also assume that although the join operator allocates 1 buffer for the output, it does not write the output to disk.

   (a) (1 point) What is the I/O cost of a grace hash join? Assume the join has 11 buffer pages of memory available.

   > **Solution:** After pass 1, Q has 10 partitions of 10 pages each. There are only 9 pages for the hash table, so it needs to do another pass over the data. Each pass is $2 * ([Q] + [P])$ and the last pass (build and probe) does not write to disk.
   > So, total I/O cost is: $5 * ([Q] + [P]) = 1300 * 5 = 6500$ I/Os.

   (b) (1 point) If we use a hybrid hash join, how many pages will be allocated for the hash table in the first phase assuming a fill factor of 0.9? Also assume the join has 32 buffer pages of memory available.

   > **Solution:**
   > $$k = (100 - (32 - 2) * 0.9)/(32 - .9 - 2) = ceil(73/29.1) = 3$$
   > number of pages for the hash table = B-k-2 = 32-3-2 = 27

(c) (3 points) How many I/Os will the first pass of the hybrid hash join incur?

Solution: 2275 I/Os.
To read in P and Q, it will take 1200 + 100 = 1300 I/Os. Roughly, 1/4 of the data will hash into memory, and the remainder into the 3 spilled partitions. So, 75 IOs to write 3 Q runs, and 900 to write 3 P runs.
The total cost of the first phase is 1300 + 900 + 75 = 2275 I/Os.

Or —

It starts with 1300 IOs. If they used the method taught in section, .73 of the buffers will spill to disk. So, 73 IOs to write 3 Q runs and 876 to write 3 R runs. Total = 2249 I/Os.

(d) (3 points) How many I/Os will the second pass of the hybrid hash join incur?

Solution: 975 I/Os
To read in and join the remaining partitions on disk is $3 * 25 = 75$ IOs for Q, and $3 * 300 = 900$ for P. Thus, the total is 975 I/Os.

Or —

If we consider the method taught in section, it will be the time to process the remaining partitions on disk. This is $876 + 73 = 949$ I/Os.

(e) (2 points) Suppose we have an index on Q with a compound key whose prefix is the join key. It is an alternative 2 clustered index with 40 leaf pages total. Assume that the upper levels of the tree are cached with a few spare (extra) buffer pages in the cache. Also assume that the query needs no additional columns from Q. Assume that the join key is a primary key in Q.
How many I/Os does INLJ take?

Solution: This will be an index only plan. Each record in P will fetch a leaf page. There are $1200 * 35 = 42,000$ records. So, this will take $1200 + 42,000 = 43200$ I/Os.

2 Parallel Queries (8 points)

The Jedi Order decides to create a database tracking all the Fighter Pilots, Sith Lords, and Jedi Masters throughout history. The order creates the following relations:

- `Jedi` (name, planet, padawan)
- `Sith` (name, planet, apprentice)
- `Pilot`(name, droid, plane, num_fighters_destroyed)

And they are partitioned across 3 machines, each with B pages of memory in the following manner (note that the Jedi Order –mistakenly–assumes a uniform distribution for all values when partitioning):

- `Jedi`: Hash-partitioned on padawan over machines 1, 2, 3.
- `Sith`: Hash-partitioned on apprentice over machines 1, 2, 3.
- `Pilot`: Range-partitioned on num_fighters_destroyed over machines 1, 2, 3.

1. (1 point) The Jedi Order realizes its pilots are woefully inexperienced - 80% of them have num_fighters_destroyed = 0, because they have never had any combat experience! Let us say that we wish to do a parallel scan on this table. Let [P] denote the number of pages in `Pilot` and let T denote the time taken for one I/O. How long will it take for a parallel scan to complete?

   A. 1/3*[P]*T

   B. [P]*T

   C. 0.8 * [P] * T

   D. 0.2*[P]*T

   Solution: Because 80% of records will end up on the same machine, the longest time taken for one of the machines to do a scan on its portion of the table will be 0.8 * [P] * T. Then, this is the bottleneck, so it will take 0.8 * [P] * T time for the scan to complete

   The order knows that many of the Sith used to be Jedi, and so they decide to perform a hash-based join on the `Jedi` and `Sith` tables to find who exactly is both Jedi and Sith. Assume that we join **only** on the name field, that we are using perfect hash functions and that the distribution of name values is not skewed for both `Jedi` and `Sith`. Let [J] be the number of pages in `Jedi` and [S] be the number of pages in `Sith`, with $[S] < [J]$.

2. (1.5 points) Suppose we first wanted to remove duplicate entries in `Sith` using hashing. What is the number of passes required to re-partition `Sith` across all 3 machines and de-duplicate the Sith records? Include the shuffle phase in your calculations. Assume we are using the *optimized* variant of parallel hashing in which shuffling the records is pipelined into the hashing operator. Assume the operator has B buffer pages of available memory.

   A. $2 + \lceil \log_{B-1}(\lceil \frac{[S]}{B} \rceil) \rceil$

   B. $2 + \lceil \log_{B-1}(\lceil \frac{[S]}{3B} \rceil) \rceil$

   C. $1 + \lceil \log_{B-1}(\lceil \frac{[S]}{B} \rceil) \rceil$

   D. $1 + \lceil \log_{B-1}(\lceil \frac{[S]}{3B} \rceil) \rceil$

Solution: In the **optimized** version each machine will get [S]/3 pages. If $S/3$ is $\leq B$, we are done with only 1 pass. Otherwise, we apply the single-machine hashing formula to a relation with $\frac{[S]}{3B}$ pages to then get our answer of $1 + \lceil \log_{B-1}(\lceil \frac{[S]}{3B} \rceil) \rceil$

Now assume that the order introduces a new table called `padawans`, which tracks apprentice Jedi who aspire to (but are not yet) be Knights. Each Padawan has one master who trains them. The schema for padawans is as follows:

```
padawans (name, age, master)
```

Assume this table is round-robin partitioned across the 3 machines. Now, assume that **60% of all padawans in the table have "Yoda" as their master**. We wish to run the following query on the table:

```
SELECT * FROM padawans WHERE master = "Yoda"
```

3. (1 point) What is the time taken for the above query to complete, if each machine has B buffer pages and `padawans` has [P] pages in total? Let T denote the time taken for 1 I/O.

   A. $\frac{1}{3}$*[P]*T

   B. 0.6*[P]*T

   C. [P]*T

   D. 0.6*$\frac{[P]}{B}$*T

Solution: Because this is round-robin partitioning and we are only dealing with a single table lookup, we can just run the query locally on each of the 3 machines and concatenate the results. Key skew will not be an issue because round-robin partitioning evenly distributes records across machines irrespective of key skew.

4. (1 point) If we wanted to sort padawans by the age key, whose values are uniformly distributed, what partitioning scheme would result in the fewest I/Os?

   A. Hash Partitioning on age

   B. Hash Partitioning on name

   C. Range Partitioning on name

   D. Range Partitioning on age

Solution: Range partitioning will be the best for this, because we can individually sort on each of the machines and concatenate the runs together at the end by appending one onto the other instead of having to merge them like in another sorting pass.

5. (1.5 points) Suppose we have time series of all monitoring events from all our fighter ships for the last 2 years:

```
events(event_time, event_type, duration, ship_id, pilot_name, pilot_rating)
```

These records tell us what is going on with our fighter fleet. To keep the fleet fighting aggressively, our engineers are constantly running queries over the latest data and for a specific event – the low fuel alert. Which partitioning scheme and physical design will achieve the best average latency if we have many engineers querying concurrently ?

A. Hash partition on `event_type` and clustered index on `event_time`

B. Range partition on `event_time` and unclustered index on `event_type`

C. Hash partition on `(ship_id, pilot_name)` and clustered index on `(event_time, event_type)`

D. Range partition on `duration` and unclustered index on `event_time`

Solution: The queries will be spread across all machines, and they will quickly find the latest data.

6. (2 points) True or False. Mark all of the true answers and leave all of the false ones blank.

A. If we partition data using the round robin scheme and then perform some operations on the data, every machine is guaranteed to have the same number of records $\pm$ 1 record.

Solution: False, deletes could remove from mainly one machine and we did not say they had to stay balanced

B. Range partitioning keeps data stored in sorted order on each machine.

Solution: False, data is partitioned according to range, but not stored in sorted order

C. A join over two tables that requires an asymmetric shuffle will incur fewer network I/Os than one that requires a symmetric shuffle.

Solution: True

D. If data is partitioned using round robin, when searching for a record, we have to perform broadcast lookup to only a subset of the machines.

Solution: False, must do it to all machines

3 Query Optimization (20 points)

For this section, remember that **the height of a one level tree is defined to be 0, the height of a tree with two levels is defined to be 1, and so on.**

Suppose the System R assumptions about uniformity and independence from lecture hold. Assume that costs are estimated as a number of I/Os, without differentiating random and sequential I/O cost. Consider the following relational schema:

```
Item (iid, name, price, popularity)
Store (sid, zip, city)
Sales (iid, sid, quantity)
```

Consider the following query:

```
SELECT name
FROM Item
WHERE price >= 100 AND price < 200
```

Suppose we have the following statistics on table Item:

tuples: 5000, **tuples/page:** 100, **total pages:** 50, **price range:** $0 - 4999$

1. (1 point) What is the estimate for the number of records output from this query?

   Solution: The selectivity factor for the prices in the range [100,200) is $100/5000 = 1/50$. So the number of matching tuples in the table becomes $100/5000 \times 5000 = 100$ tuples.

2. (1 point) If we build an unclustered alternative 2 B+-tree index of height 2 on Item.price and use an index scan to process this query, what is the number of IOs incurred? Assume that that we only need to read at most one leaf node and assume that we can cache the index.

   Solution: If we use the unclustered B+ tree index, it takes one IO to retrieve each tuple. Thus the total IO cost is $3 + 100 = 103$ I/Os.

3. (1 point) True or False: If we build a clustered alternative 2 B+-tree index of height 2 on Item.price and use an index scan to process this query, the number of IOs is less than the number of IOs required for a sequential scan. Assume that that we only need to read at most one leaf node and assume that we can cache the index.

   A. True

   B. False

   Solution: True. The number of matching tuples for the predicate is still 100 as above. However, since we are using a clustered B+ tree index, it takes 1 IO to retrieve a page of sequential tuples. Thus the total IO cost is $3 + 100/100 = 4$ IOs, which is less than the 50 IOs required for a sequential table scan.

Consider the following query with a histogram on Sales.quantity:

```
SELECT name, price
FROM Item, Store, Sales
WHERE Item.iid = Sales.iid AND Store.sid = Sales.sid AND Sales.quantity >= 100
ORDER BY price
```

0-100	100-200	200-300	300-400	400-500
15%	20%	20%	15%	30%

Figure 1: Histogram on Sales.quantity.

4. (1 point) True or False: Assume a sale always has a corresponding item and store. Applying a filter using the histogram on Sales.quantity before joining always yields a cheaper plan for this query.

   A. True

   B. False

   Solution: If Sales is the inner table when it is joined, then pushing the select down would not reduce the I/O cost unless we materialized the results.

5. (1 point) Based on the histogram on Sales.quantity, what is the selectivity for Sales.quantity >= 100 ? Assume that the histogram boundaries are left inclusive and right exclusive.

   Solution: $1 - .15 = .85$.

6. (1 point) True or False: The reason System R optimizer will not consider joining two tables with no join predicate (i.e. performing a Cartesian product) is because these plans will always be suboptimal.

   A. True

   B. False

   Solution: False. Sometimes a Cartesian product does result in an optimal plan. For the most part, however, it doesn't so Selinger optimizer prunes these plans to reduce search space.

7. (2 points) Suppose we have the following list of 2 table subplans and their costs. Mark the letters of the plans that will be chosen by the optimizer. Ignore interesting orders.

   A. Item $\bowtie$ Sales (2,000 IO)

   B. Sales $\bowtie$ Item (1,500 IO)

   C. Store $\bowtie$ Sales (3,000 IO)

   D. Sales $\bowtie$ Store (4,000 IO)

   Solution: Plans B and C will be chosen. Since Plans A and B both join the same sets of tables, we retain B over A because it has a cheaper IO cost. The same reasoning applies when we decide to retain Plan C over Plan D.

8. (4 points) Based on the plans that were retained in the last pass (the previous question), now suppose we have the following list of 3 table access plans and their costs. Mark the letters for all 3 table access plans that would be considered by the optimizer and write down the letter of the final query plan that will be chosen. (Cost given are cumulative for the full query. Again, ignore interesting orders in formulating your answer.)

A. (Item ⋈ Sales) ⋈ Store (2,500,000 IO)

B. (Sales ⋈ Item) ⋈ Store (2,000,000 IO)

C. Store ⋈ (Item ⋈ Sales) (1,800,000 IO)

D. Store ⋈ (Sales ⋈ Item) (1,500,000 IO)

E. (Sales ⋈ Store) ⋈ Item (4,500,000 IO)

F. (Store ⋈ Sales) ⋈ Item (3,500,000 IO)

G. Item ⋈ (Sales ⋈ Store) (3,000,000 IO)

H. Item ⋈ (Store ⋈ Sales) (2,800,000 IO)

Solution: Plans B and F will be considered (System R doesn't consider right-deep plans, and only (Store ⋈ Sales) and (Sales ⋈ Item) are retained from the last pass, eliminating all other choices). B will be picked because it is the cheapest of the two remaining plans.

9. (1 point) True or False: If we have an unclustered B+-tree index on Sales.quantity, the System R optimizer may consider the index scan due to interesting order even if a sequential scan is cheaper.

A. True

B. False

Solution: False. since quantity is not used in the query downstream, it is not an interesting order.

Consider the above histogram for Sales.quantity and the additional histograms below on Item.price and Item.popularity. Assume that all histogram boundaries are left inclusive and right exclusive. (Simplify all your answers, and use 2 significant digits.)

0-100	100-300	300-900	900-2000	2000-4999
50%	25%	15%	7%	3%

Figure 2: Histogram on Item.price.

0-4	4-6	6-8	8-9	9-10
10%	20%	40%	25%	5%

Figure 3: Histogram on Item.popularity.

10. (2 points) What is the selectivity for the predicate:

Sales.quantity >= 225 AND Item.popularity >= 2 AND Item.popularity < 6

Solution: Using independence and uniformity in the bins: (.15 + .15 + 0.3) * (.05 + 0.20) = (.6)*(.25) = .15.

11. (2 points) What is the selectivity for the predicate:

(Item.price >= 50 and Item.price < 700) OR (Item.popularity >= 7)

Solution: Again, using uniformity and independence assumptions
the first part of disjunct: .25 + .25 + .10 = 0.60.
the Second part of disjunct: .20 + .25 + .05 = 0.5
Overall = 0.60 + 0.50 - (0.50)*(0.60) = .80

Consider the same query above. Suppose the optimizer has narrowed the candidate plans to the following two:

Plan 1: (Item $\bowtie$ Sales) $\bowtie$ Store) Estimated total IO: 2,500,000

Plan 2: (Sales $\bowtie$ Store) $\bowtie$ Item) Estimated total IO: 4,500,000

In this case, the optimizer will choose plan 1 as its estimated IO is less than that of plan 2. In the lecture, we have been making assumptions that the data distribution is uniform and the columns are independent of each other. However, these assumptions rarely hold in real life. Consequently, the actual IO might differ significantly from the estimated IO.

One way to potentially improve the plan selection is to incorporate feedback; after executing the query using plan 1 and observing the actual IO of this plan, we can feed this information back to the optimizer to let it better optimize executions of the same query in the future.

12. (1 point) Suppose we observe that the actual IO is 5,000,000 for plan 1. If we rerun the same query, which plan will the optimizer choose this time?

    A. Plan 1

    B. Plan 2

    Solution: Plan 2, since its estimated IO cost is cheaper than the actual IO cost for Plan 1.

13. (1 point) Suppose we observe that the actual IO is 2,000,000 for plan 1. If we rerun the same query, which plan will the optimizer choose this time?

    A. Plan 1

    B. Plan 2

    Solution: Plan 1, since its actual IO cost is cheaper than the estimated IO cost for Plan 2.

14. (1 point) True or False: With this feedback mechanism, the optimizer will eventually find the optimal execution plan (in terms of the actual IO, not estimated IO) after running the same query over and over again.

    A. True

    B. False

    Solution: False. Consider the case where Plan 1 has an estimated IO cost of 2,500,000 and an actual cost of 2,000,000, while Plan 2 has an estimated IO cost of 4,500,000 and an actual IO cost of 1,500,000. The optimizer will always choose Plan 1 since both its actual and estimated IO costs are lower than Plan 2's estimated IO cost, when in reality Plan 2 has a cheaper actual IO cost.

4 Transactions and Concurrency Control (12 points)

1. (1.5 points) Fill in the corresponding box on the answer sheet if True.

   A. If only the last two operations out of many in a committed transaction do not take effect, it is a violation of the Atomicity principle.

   B. Interleaving transactions is always faster than doing them one at a time.

   C. All conflict serializable are also serial schedules.

   D. Transaction A reads from relation W, and Transaction B reads from relation W concurrently. They may possibly read any or all of W. This is considered a conflict.

   E. Transaction A writes to relation W, and Transaction B writes to relation W concurrently. They may possibly write to any or all of W. This is considered a conflict.

Solution:
A is a violation of the Atomicity principle.
For B, there could be no conflicts and since interleaving has a significant overhead cost, interleaving could actually run slower.
C is false because a serial schedule is one in which each transactions runs from start to finish without any interweaving whereas conflict serializable schedules can have interweaving.
D is false because a conflict requires at least one operation to be a write and both operations here are reads.
E is a definition of a conflict

Use the schedule below for the next two questions. Assume both A and B have a value of 100 before T1 begins.

T1	R(B)	R(A)	A=A+100	B=B+A	W(A)	B=B+100	W(B)	A=A+B	W(A)	COM

2. (1 point) Suppose T1 successfully commits.
   (a) What is the final value of A?

   Solution: 600

   (b) What is the final value of B?

   Solution: 400

3. (2 points) Suppose T1 starts and its execution is **not** atomic. (You may mark none or more than one for the choices below.)
   (a) Which of the following are possible final states of A?

   A. 100

   B. 200

   C. 600

   (b) Which of the following are possible final states of B?

A. 100
B. 300
C. 400

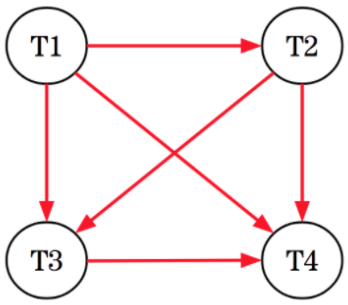
4. (4.5 points) Use the schedule below for this question. (You may mark none or more than one for the choices below.)

T1	R(B)						R(C)			R(A)	
T2		R(B)			R(D)						W(A)
T3			W(B)			R(C)					
T4				W(B)				W(C)	W(D)		

(a) What transactions is T1 pointing to in the conflict graph for the schedule above?

A. T2
B. T3
C. T4

Solution:



(b) What transactions is T2 pointing to in the dependency graph for the schedule above?

A. T1
B. T3
C. T4

(c) What transactions is T3 pointing to in the dependency graph for the schedule above?

A. T1
B. T2
C. T4

(d) What transactions is T4 pointing to in the dependency graph for the schedule above?

A. T1
B. T2
C. T3

Solution: None of these are the correct answers.

(e) True or False: The schedule above is conflict serializable.

Solution: True. There are no cycles in the dependency graph.

(f) Which of the following schedules below are conflict equivalent to the schedule above?

A. T2, T1, T4, T3

B. T2, T1, T3, T4

C. T1, T2, T3, T4

D. T1, T2, T4, T3

Solution: C is the correct choice. If you were to do a topological sort on the dependency graph, C is the only ordering which is possible. T2 depends on T1 to finish so that eliminates A and B and T4 depends on T3 to finish so D is eliminated.

5. (3 points) Consider the following schedule, with A = 90 and B = 90 at the start. Mark the statements below that are True.

T1	T2
	read(A)
	read(B)
	if (A>100) **then** B=A+B
read(B)	
read(A)	
if (B>100) **then** A=A+B	
else A=A+10	
write(A)	
	else B=B+10
	write(B)

A. The schedule is serializable.

B. The schedule is view serializable.

C. The schedule is not conflict serializable

Solution: A and C are the correct choices. In this case, A and B are never larger than 100. So there is no real conflict between T1 and T2. However, both conflict and view serializable definitions disallow it. There is a cycle between T1 and T2 because of read(B) → write(B) and read(A) → write(A), it's not conflict serializable. In a serial schedule, one of the two would read the write from the other, so it's not view serializable.