CS3223 Tutorial 6: Query Optimization & Serializability

Wk 9, Sem 2, 2022/23

Questions to be discussed: 1 & 4

1. Consider a database with the following two relations where the key attributes are shown underlined:

   - R (a, b, c, d)
   - T (h, i, j)

   Assume the following for this question.

   1. Relation `R` contains 10,000 pages with each page containing 30 records.
   2. Relation `T` contains 8,000 pages with each page containing 100 records.
   3. There are three unclustered indexes on relation `R`:
      - I_b: a B^+-tree index on (b) with at most 100 entries in each leaf page.
      - I_c: a B^+-tree index on (c) with at most 100 entries in each leaf page.
      - I_{bc}: a B^+-tree index on (b, c) with at most 50 entries in each leaf page.
   4. There are three unclustered indexes on relation `T`:
      - I_i: a B^+-tree index on (i) with at most 200 entries in each leaf page.
      - I_j: a B^+-tree index on (j) with at most 200 entries in each leaf page.
      - I_{ij}: a B^+-tree index on (i, j) with at most 100 entries in each leaf page.
   5. Each of the indexes has two levels of internal nodes.
   6. Only 10% of `R` records satisfy the condition "b > 20".
   7. Only 5% of `R` records satisfy the condition "c = 100".
   8. Only 1% of `R` records satisfy both the conditions "b > 20" and "c = 100".
   9. Only 5% of `T` records satisfy the condition "i > 50".
   10. Only 5% of `T` records satisfy the condition "j > 30".
   11. Only 4% of `T` records satisfy both the conditions "i > 50" and "j > 30".
   12. The cost metric to use is the number of page I/Os. Ignore the cost of writing out the final result.
   13. There are 25 buffer pages available.
   14. The database system supports only four join algorithms (Block Nested-loop Join, Indexed Nested-loop Join, Optimized Sort-Merge Join, and Grace Hash Join), and supports only the hash-based algorithm for set intersections.

   Consider the following three queries.

```
Q1: SELECT *
    FROM   R
    WHERE  b > 20
    AND    c = 100
```

```
Q2: SELECT *
    FROM   T
    WHERE  i > 50
    AND    j > 30
```

```
Q3: SELECT *
    FROM   R, T
    WHERE  R.d = T.h
    AND    R.b > 20
    AND    R.c = 100
    AND    T.i > 50
    AND    T.j > 30
```

   Answer the following three questions:

   (a) What is the least cost plan for query $Q1$? What is its cost?

   (b) What is the least cost plan for query $Q2$? What is its cost?

   (c) What is the least cost plan for query $Q3$? What is its cost?

Solution:

- $||R|| = 10,000 \times 30 = 300{,}000$
- Number of leaf pages in $I_b = \frac{300,000}{100} = 3000$.
- Number of leaf pages in $I_c = \frac{300,000}{100} = 3000$.
- Number of leaf pages in $I_{bc} = \frac{300,000}{50} = 6000$.
- $||\sigma_{b>20}(R)|| = 0.1 \times ||R|| = 30,000$. $|\sigma_{b>20}(R)| = \frac{30,000}{30} = 1000$
- $||\sigma_{c=100}(R)|| = 0.05 \times ||R|| = 15,000$. $|\sigma_{c=100}(R)| = \frac{15,000}{30} = 500$
- $||\sigma_{(b>20)\wedge(c=100)}(R)|| = 0.01 \times ||R|| = 3,000$. $|\sigma_{(b>20)\wedge(c=100)}(R)| = \frac{3000}{30} = 100$
- $||T|| = 8,000 \times 100 = 800{,}000$
- Number of leaf pages in $I_i = \frac{800,000}{200} = 4000$.
- Number of leaf pages in $I_j = \frac{800,000}{200} = 4000$.
- Number of leaf pages in $I_{ij} = \frac{800,000}{100} = 8000$.
- $||\sigma_{i>50}(T)|| = 0.05 \times ||T|| = 40,000$. $|\sigma_{i>50}(T)| = \frac{40,000}{100} = 400$
- $||\sigma_{j>30}(T)|| = 0.05 \times ||T|| = 40,000$. $|\sigma_{j>30}(T)| = \frac{40,000}{100} = 400$
- $||\sigma_{(i>50)\wedge(j>20)}(T)|| = 0.04 \times ||T|| = 32,000$. $|\sigma_{(i>50)\wedge(j>20)}(T)| = \frac{32000}{100} = 320$

(a) Plan P1: Table scan. Cost(P1) = $|R|$ = 10,000.

Plan P2: Index scan with I_b. Cost(P2) = $2 + (0.1 \times 3000) + 30,000 = 30,302$.

Plan P3: Index scan with I_c. Cost(P3) = $2 + (0.05 \times 3000) + 15,000 = 15,152$.

Plan P4: Index scan with $I_{b,c}$. Cost(P4) = $2 + (0.1 \times 6000) + 3,000 = 3602$.

Plan P5: Index intersection of I_b & I_c using hash-based approach.

- Cost(P5) = Cost to partition matching leaf entries in I_b + Cost to partition matching leaf entries in I_c + Cost to intersect matching leaf entries in I_b & I_c + Cost to retrieve matching data records
- Cost to scan I_b: $2 + (0.1 \times 3000) = 302$
  - Number of pages to store matching data entries, $P_b = \frac{30,000}{100} = 300$
- Cost to scan I_c: $2 + (0.05 \times 3000) = 152$
  - Number of pages to store matching data entries, $P_c = \frac{15,000}{100} = 150$
- Since $P_c < P_b$, we use P_c as the build relation.
- Size of build partition = $\lceil \frac{P_c}{B-1} \rceil = \lceil \frac{150}{24} \rceil = 7$; thus, no repartitioning is required.
- Cost to partition matching leaf entries in I_b = Cost to scan I_b + Cost to write partitions = 302 + 300 = 602
- Cost to partition matching leaf entries in I_c = Cost to scan I_c + Cost to write partitions = 152 + 150 = 302
- Cost to intersect matching leaf entries in I_b & I_c = 300 + 150 = 450
- Cost to retrieve matching data records = $||\sigma_{(b>20)\wedge(c=100)}(R)|| = 3000$
- Cost(P5) = 602 + 302 + 450 + 3000 = 4354

The best plan is P4 with a cost of 3602.

(b) Plan P1: Table scan. Cost(P1) = 8,000.

Plan P2: Index scan with I_i. Cost(P2) = $2 + (0.05 \times 4000) + 40,000 = 40202$.

Plan P3: Index scan with I_j. Cost(P3) = $2 + (0.05 \times 4000) + 40,000 = 40202$.

Plan P4: Index scan with $I_{i,j}$. Cost(P4) = $2 + (0.05 \times 8000) + 32,000 = 32402$.

Plan P5: Index intersection of I_i & I_j using hash-based approach.

- Cost(P5) = Cost to partition matching leaf entries in I_i + Cost to partition matching leaf entries in I_j + Cost to intersect matching leaf entries in I_i & I_j + Cost to retrieve matching data records
- Cost to scan I_i: $2 + (0.05 \times 4000) = 202$
  - Number of pages to store matching data entries, $P_i = \frac{40,000}{200} = 200$
- Cost to scan I_j: $2 + (0.05 \times 4000) = 202$
  - Number of pages to store matching data entries, $P_j = \frac{40,000}{200} = 200$
- We use P_i as the build relation.
- Size of build partition = $\lceil \frac{P_i}{B-1} \rceil = \lceil \frac{200}{24} \rceil = 7$; thus, no repartitioning is required.
- Cost to partition matching leaf entries in I_i = Cost to scan I_i + Cost to write partitions = 202 + 200 = 402
- Cost to partition matching leaf entries in I_j = Cost to scan I_j + Cost to write partitions = 202 + 200 = 402
- Cost to intersect matching leaf entries in I_i & I_j = 200 + 200 = 400
- Cost to retrieve matching data records = $||\sigma_{(i>50)\wedge(j>20)}(T)|| = 32,000$.
- Cost(P5) = 402 + 402 + 400 + 32,000 = 33,204

Therefore, the best plan is P1 with a cost of 8000.

(c) **Plan P1 = Block nested-loop join with R as outer relation.**

The number of times that the inner tuples need to be accessed is given by $\lceil \frac{|\sigma_{(b>20)\wedge(c=100)}(R)|}{B-2} \rceil$ = $\lceil \frac{100}{23} \rceil = 5$. Based on part (b), the best plan to access the inner tuples is a table scan of T. We need to determine whether it is more more efficient to materialize $\sigma_{(i>50)\wedge(j>30)}(T)$.

- Without materialization, the cost to access the inner tuples = $5 \times |T| = 5 \times 8000 = 40,000$.
- With materialization, the cost to access the inner tuples = Cost to materialize + Cost to access materialized relation. Cost to materialize $\sigma_{(i>50)\wedge(j>30)}(T)$ = Cost of table scan on T + Cost to write materialized relation = 8000 + 320 = 8320. Cost to access materialized relation = $5 \times 320 = 1600$. Therefore, with materialization, the cost to access the inner tuples = 8320 + 1600 = 9920.

Therefore, we should materialize $\sigma_{(i>50)\wedge(j>30)}(T)$.

Cost(P1) = Cost to materialize $\sigma_{(i>50)\wedge(j>30)}(T)$ + Cost to access outer relation + Cost to access materialized relation.

Based on part (a), the best plan to access relation R is an index scan with $I_{b,c}$. Therefore, cost to access outer relation = 3602.

Therefore, Cost(P1) = 3602 + 8320 + 1600 = 12,622.

Plan P2 = Sort-merge join with R as outer relation.

Number of initial sorted runs for R = $\lceil \frac{100}{25} \rceil = 4$.

Number of initial sorted runs for T = $\lceil \frac{320}{25} \rceil = 13$.

There is sufficient buffer pages to merge and join the operands in a single pass.

Based on parts (a) and (b), we access R using an index scan with $I_{b,c}$ and access T using a table scan.

Cost of Pass 0 for R = Cost of I_{bc} scan on R + Cost to write sorted runs = 3602 + 100 = 3702.

Cost of Pass 0 for T = Cost of table scan of T + Cost to write sorted runs = 8000 + 320 = 8320.

Since each R-tuple joins with at most one T-tuple, and R is the outer relation, any re-scanning of T will not incur any additional disk I/O. Therefore, by applying the optimized sort-merge join, the cost of merge & join of initial sorted runs = 100 + 320 = 420.

Cost(P2) = 3702 + 8320 + 420 = 12,442.

Plan P3 = Hash join with R as build relation.

Size of R partition = $\frac{|\sigma_{(b>20)\wedge(c=100)}(R)|}{B-1} = \frac{100}{24} = 5$; thus, no repartitioning is required.

Cost(P3) = (cost to partition R) + (cost to partition T) + (cost of probe phase)

Based on parts (a) and (b), we access R using an index scan with $I_{b,c}$ and access T using a table scan.

Cost to partition R = (cost of I_{bc} scan on R) + (cost to write partitions) = 3602 + 100 = 3702.

Cost to partition T = (cost of table scan on T) + (cost to write partitions) = 8000 + 320 = 8320.

Cost for probe phase = 100 + 320 = 420.

Therefore, Cost(P3) = 3702 + 8320 + 420 = 12,442.

Plan P4 = Block nested-loop join with R as inner relation.

Since $|\sigma_{(b>20)\wedge(c=100)}(R)| < |\sigma_{(i>50)\wedge(j>20)}(T)|$, Cost(P4) > Cost(P1).

Plan P5 = Sort-merge join with R as inner relation.

Since each tuple in R joins with at most one tuple in T, Cost(P5) $\geq$ Cost(P2).

Plan P6 = Hash join with T as build relation.

Since $|\sigma_{(b>20)\wedge(c=100)}(R)| < |\sigma_{(i>50)\wedge(j>20)}(T)|$, Cost(P6) $\geq$ Cost(P3).

Therefore, the least cost plan is P2 (sort-merge join) or P3 (hash join) with a cost of 12,442.

2. Consider a relation $R(\underline{a}, b, c)$, where the domains of all the attributes are positive integers. Assume that $||R|| = 10000$, $||\pi_b(R)|| = 100$, and $||\pi_c(R)|| = 20$.

   Estimate the result size for each of the following queries.

   (a) SELECT * FROM R WHERE $b = 10$

   (b) SELECT * FROM R WHERE $(b \geq 20)$ AND $(b < 40)$

   (c) SELECT * FROM R WHERE $b \neq 7$

   (d) SELECT * FROM R WHERE $(b = 20)$ AND $(c = 40)$

   (e) SELECT * FROM R WHERE $(b = 20)$ OR $(c = 40)$

Solution:

(a) $\frac{1}{100} \times 10000 = 100$

(b) $\frac{40-20}{100} \times 10000 = 2000$

(c) $(1 - \frac{1}{100}) \times 10000 = 9900$

(d) $\frac{1}{100} \times \frac{1}{20} \times 10000 = 5$

(e) $(1 - (1 - \frac{1}{100}) \times (1 - \frac{1}{20})) \times 10000 = (1 - \frac{99}{100}\frac{19}{20}) \times 10000 = 595$

3. Consider a database with the following three relations:

   - $R(a, b, c)$ with $||R|| = 200$, $||\pi_b(R)|| = 20$, and $||\pi_c(R)|| = 50$
   - $S(d, e, b)$ with $||S|| = 800$ and $||\pi_b(S)|| = 40$
   - $T(f, g, c)$ with $||T|| = 500$ and $||\pi_c(T)|| = 100$

   Estimate the result cardinality for each of the following queries:

   (a) Q_1: SELECT * FROM R JOIN S ON R.b = S.b

   (b) Q_2: SELECT * FROM R JOIN S ON R.b = S.b JOIN T ON R.c = T.c

Solution:

(a) $||Q_1|| = ||R|| \times ||S|| \times \frac{1}{\max\{||\pi_b(R)||, ||\pi_b(S)||\}} = \frac{200 \times 800}{40} = 4,000$

(b) $||Q_2|| = ||R|| \times ||S|| \times ||T|| \times \frac{1}{\max\{||\pi_b(R)||, ||\pi_b(S)||\}} \times \frac{1}{\max\{||\pi_c(R)||, ||\pi_c(T)||\}} = \frac{200 \times 800 \times 500}{40 \times 100} = 20,000$

4. Consider a relation R with $||R|| = 121$ and an attribute A with $||\pi_A(R)|| = 20$. The actual distribution of attribute A is shown below.

Value of A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
# of tuples	1	15	6	8	2	3	0	1	20	3	0	10	6	8	12	7	0	8	5	6

(a) Construct an equidepth histogram H_3 with 3 buckets.

(b) Estimate the size for each of the following queries using H_3.
   i. Q_1: SELECT * FROM R WHERE A = 5
   ii. Q_2: SELECT * FROM R WHERE A = 8
   iii. Q_3: SELECT * FROM R WHERE A $\geq$ 6 AND A $\leq$ 17

(c) Construct an equidepth histogram H_3' with 3 buckets and top-2 MCV.

(d) Repeat part (b) using H_3'.

(e) Construct an equidepth histogram H_5 with 5 buckets.

(f) Repeat part (b) using H_5.

Solution: We have $||Q_1|| = 3$, $||Q_2|| = 20$, and $||Q_3|| = 75$.

(a) There're multiple possible equidepth histograms for this question. Here's one possibility.

Bucket Num.	Value Range	Num. of Tuples
1	[0-8]	1+15+6+8+2+3+0+1+4=40
2	[8-13]	16+3+0+10+6+5=40
3	[13-19]	3+12+7+0+8+5+6=41

(b) i. $\frac{1}{9} \times 40 \approx 4.44$
   ii. $(\frac{1}{9} \times 40) + (\frac{1}{6} \times 40) \approx 11.11$
   iii. $(\frac{3}{9} \times 40) + 40 + (\frac{5}{7} \times 41) \approx 82.62$

(c) Similarly here, there're multiple possible equidepth histograms for this question. Here's one possibility.

Bucket Num.	Value Range	Num. of Tuples
1	[0-11]	1+0+6+8+2+3+0+1+0+3+0+4=28
2	[11-14]	6+6+8+9=29
3	[14-19]	3+7+0+8+5+6=29

Value	Num. of Tuples
8	20
1	15

(d) i. $\frac{1}{12-2} \times 28 = 2.8$
   ii. 20
   iii. $20 + (\frac{6-1}{12-2} \times 28) + 29 + (\frac{4}{6} \times 29) \approx 82.33$

(e) Here's one possible 5-bucket equidepth histogram.

Bucket Num.	Value Range	Num. of Tuples
1	[0-3]	1+15+6+2=24
2	[3-8]	6+2+3+0+1+12=24
3	[8-12]	8+3+0+10+3=24
4	[12-15]	3+8+12+1=24
5	[15-19]	6+0+8+5+6=25

(f) i. $\frac{24}{6} = 4$
   ii. $\frac{24}{6} + \frac{24}{5} = 8.8$
   iii. $(\frac{3}{6} \times 24) + 24 + 24 + (\frac{3}{5} \times 25) = 75$

5. (Exercise 17.2, R&G) For each of the following schedules, state whether is it view/conflict serializable.
   (a) R_1(X), R_2(X), W_1(X), W_2(X), $Commit_1$, $Commit_2$
   (b) W_1(X), R_2(Y), R_1(Y), R_2(X), $Commit_1$, $Commit_2$
   (c) R_1(X), R_2(Y), W_3(X), R_2(X), R_1(Y), $Commit_1$, $Commit_2$, $Commit_3$
   (d) R_1(X), R_1(Y), W_1(X), R_2(Y), W_3(Y), W_1(X), R_2(Y), $Commit_1$, $Commit_2$, $Commit_3$
   (e) R_1(X), W_2(X), W_1(X), $Commit_2$, $Commit_1$
   (f) W_1(X), R_2(X), W_1(X), $Commit_2$, $Commit_1$
   (g) R_2(X), W_3(X), $Commit_3$, W_1(Y), $Commit_1$, R_2(Y), W_2(Z), $Commit_2$
   (h) R_1(X), W_2(X), $Commit_2$, W_1(X), $Commit_1$, R_3(X), $Commit_3$
   (i) R_1(X), W_2(X), W_1(X), R_3(X), $Commit_1$, $Commit_2$, $Commit_3$
   (j) R_1(X), R_2(Y), W_3(X), W_3(Z), R_2(X), R_1(Y), W_1(Z), W_2(Z), $Commit_1$, $Commit_2$, $Commit_3$

Solution:

To determine whether a schedule S is CSS, we construct CSG(S) and determine whether CSG(S) is cyclic: S is CSS iff CSG(S) is acyclic. Clearly, if S is CSS, then S is also VSS.

To determine whether a schedule S is VSS, we can construct a directed graph (denoted by VSG(S)) to capture the read-from and final-write relations among the transactions. Each node in VSG(S) represents a transaction with the following edges: (1) (T_j, T_i) if T_i reads from T_j, (2) (T_j, T_i) if both T_i & T_j update the same object O & T_i performs the final write on O, and (3) (T_j, T_i) if T_j read some object O from the initial database & T_i update object O. If VSG(S) is cyclic, then S is not VSS. If VSG(S) is acyclic, then S is VSS iff there exists a serial schedule produced from a topological ordering of VSG(S) that is view equivalent to S.

(a) **not view serializable**, **not conflict serializable**

(b) **view serializable**, **conflict serializable**

(c) **view serializable**, **conflict serializable**

(d) **not view serializable**, **not conflict serializable**,

(e) **not view serializable**, **not conflict serializable**.

(f) **not view serializable**, **not conflict serializable**

(g) **view serializable**, **conflict serializable**

(h) **not view serializable**, **not conflict serializable**

(i) **not view serializable**, **not conflict serializable**

(j) **view serializable**, **not conflict serializable**

6. Prove that a conflict serializable schedule is also a view serializable schedule.

Solution: Let S be a conflict serializable schedule. Thus, there exists a serial schedule S' that is conflict equivalent to S. We establish that S and S' must be view equivalent by contradiction. Suppose that S and S' are not view equivalent. This implies that S and S' differ in some (1) final write or (2) read-from relationship.

Consider case (1). Suppose that $W_i(A)$ is the final write on A in S and $W_j(A)$ is the final write on A in S', $i \neq j$; i.e.,

$$S : \cdots W_j(A) \cdots W_i(A) \cdots$$
$$S' : \cdots W_i(A) \cdots W_j(A) \cdots$$

However, this contradicts the fact that S and S' are conflict equivalent since the pair of conflicting actions, $W_i(A)$ and $W_j(A)$, are ordered differently in S and S'. Hence, S and S' must agree on the same final writes.

Consider case (2). Suppose that T_k reads A from T_i in S, and T_k reads A from T_j in S', $i \neq j$; i.e.,

$$S : \cdots W_i(A) \cdots R_k(A) \cdots$$
$$S' : \cdots W_j(A) \cdots R_k(A) \cdots$$

Since $W_i(A)$ precedes $R_k(A)$ in S, and S and S' are conflict equivalent, we must have $W_i(A)$ precedes $R_k(A)$ in S'. By a similar argument, we must have $W_j(A)$ precedes $R_k(A)$ in S. Therefore, we have the following:

$$S : \cdots W_j(A) \cdots W_i(A) \cdots R_k(A) \cdots$$
$$S' : \cdots W_i(A) \cdots W_j(A) \cdots R_k(A) \cdots$$

However, this contradicts the fact that S and S' are conflict equivalent since the pair of conflicting actions, $W_i(A)$ and $W_j(A)$, are ordered differently in S and S'. Hence, S and S' must agree on the same read-from relationships.

Therefore, S and S' must be view serializable if they are conflict serializable.

7. Prove that a schedule is conflict serializable if and only if its conflict serializability graph is acyclic.

Solution:

Let S be a schedule.

Only if. Assume S is CSS. By definition, there exists a serial schedule S' that is conflict equivalent to S. Suppose that CSG(S) is cyclic. Since CSG(S') = CSG(S), therefore CSG(S') is also cyclic; i.e., there exists some path in CSG(S') from some T_i to T_i. But this is a contradiction since S' is a serial schedule. Therefore, if S is CSS, then CSG(S) must be acyclic.

If. Assume that CSG(S) is acyclic. Let S' be a serial schedule obtained from a topological ordering of CSG(S).

We first now show that every pair of conflicting actions in S is also in S'. Consider a pair of conflicting actions (a_i, a_j) in S, where a_k denotes some action in Xact T_k and a_i precedes a_j in S. Thus, there is an edge in CSG(S) representing (a_i, a_j). Therefore, T_i must precede T_j in S' since S' is derived from a topological ordering of CSG(S). Thus, the pair of conflicting actions (a_i, a_j) is also in S'.

We next now show that every pair of conflicting actions in S' is also in S. Consider a pair of conflicting actions (a_i, a_j) in S'. Since S' is obtained from a topological ordering of CSG(S), there must be an edge from T_i to T_j. So the pair of conflicting actions (a_i, a_j) is also in S. Therefore, S is conflict equivalent to S' (i.e., S is CSS).

8. Prove that a view serializable schedule without any blind write is also a conflict serializable schedule.

Solution: Given a schedule S, we use $(A_1, A_2, \cdots, A_k)$ to denote a subsequence of k actions in S.

Since S is VSS, there must exist a serial schedule S' that is view equivalent to S. We show that the set of conflicting actions in S and S are equivalent.

Case 1: Assume that we have $(W_i(x), W_j(x))$ in S and $(W_j(x), W_i(x))$ in S'. Since there are no blind writes in S, we must have $(R_i(x), W_i(x), W_j(x))$ in S and $(W_j(x), R_i(x), W_i(x))$ in S'.

Since the read-from relations are equivalent in S and S', there must exist a $W_k(x)$ such that $(W_k(x), R_i(x), W_i(x), W_j(x))$ in S and $(W_j(x), W_k(x), R_i(x), W_i(x))$ in S'. Thus, whenever we have $(W_i(x), W_j(x))$ in S and $(W_j(x), W_i(x))$ in S', we must have $(W_k(x), R_i(x), W_i(x), W_j(x))$ in S and $(W_j(x), W_k(x), R_i(x), W_i(x))$ in S'. But now we have $(W_k(x), W_j(x))$ in S and $(W_j(x), W_k(x))$ in S' which implies that there must exist a $W_p(x)$ such that $(W_p(x), R_k(x), W_k(x), W_j(x))$ in S and $(W_j(x), W_p(x), R_k(x), W_k(x))$ in S'. This leads to an infinite number of writes! Therefore, the set of write-write conflicts are equivalent in S and S'.

Case 2: Suppose we have $(R_i(x), W_j(x))$ in S and $(W_j(x), R_i(x))$ in S'. Since S is VSS, the read-from relations must be the same; i.e., we must have $(W_k(x), R_i(x), W_j(x))$ in S and $(W_j(x), W_k(x), R_i(x))$ in S'. But by case 1, this is a contradiction. Therefore, the set of write-read/read-write conflicts are equivalent in S and S'.

Hence S must also be CSS.