Database Systems, CSCI 4380-01
Homework # 9
Due Wednesday December 12, 2018 at 11:59 PM

Introduction.

This homework is worth 2.5% of your total grade. If you choose to skip it, the final exam will be worth 2.5% more. If you complete all the homeworks after Exam #2, there will be a special bonus in terms of your final grade computation. Additionally, doing this work now is essential to doing well in the final exam.

This homework requires no programming, only pen/pencil on paper computations. So you will submit your answers as a PDF file on GRADESCOPE.

Question 1. You are given the following two schedules.

(a) list all conflicts and draw the conflict graph.

(b) discuss whether the schedule is serializable or not with a one sentence explanation of why. If it is serializable, find an equivalent serial schedule.

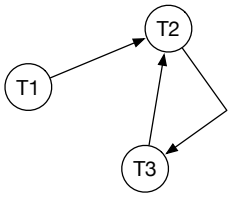
$$
\begin{array}{lcl}
S1 & : & r_1(X)\ r_1(Y)\ w_2(Y)\ w_2(Z)\ r_3(Z)\ w_3(K)\ r_2(K)\ w_2(L)\ w_1(X) \\
S2 & : & r_1(X)\ w_2(X)\ w_1(Y)\ r_3(Y)\ w_3(Z)\ r_2(Z)\ r_3(W)\ w_4(W)\ w_2(Z)\ r_4(W)
\end{array}
$$

Answer (a).

Conflicts for S1:

$$
\begin{array}{r}
r_1(Y)\ w_2(Y) \\
w_2(Z)\ r_3(Z) \\
w_3(K)\ r_2(K)
\end{array}
$$

Conflict graph (see below):



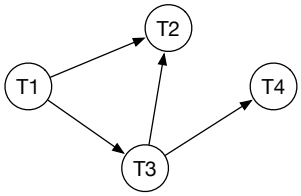
Since there is a cycle between T2 and T3, this schedule is not serializable.

Answer (b).

Conflicts for S2:

$$r_1(X)\ w_2(X)$$
$$w_1(Y)\ r_3(Y)$$
$$w_3(Z)\ r_2(Z)$$
$$w_3(Z)\ w_2(Z)$$
$$r_3(W)\ w_4(W)$$

Conflict graph (see below):



This graph has no cycle, hence this schedule is serializable. The following is a serial schedule that is equivalent to this schedule: T1, T3, T2, T4 or T1, T3, T4, T2.

Question 2. You are given the following contents of the log and data pages after a crash. Based on this, discuss which transactions must be aborted, which actions must be undone and which actions must be redone. You can choose to REDO all actions or only those actions by committed transactions. However, you must also REDO all CLR actions as well. You do not need to worry about which new log items are written for the recovery.

First list which transactions need to be aborted. Then, go through each potential redo/undo operation in the order attempted for recovery (forward for REDO and backwards for UNDO), read the data page into memory, discuss what is the last stored change for that page and whether it needs to be redone or undone.

Each update operation is listed as `PAGEID UPDATE OLDVALUE NEWVALUE`.

For each transaction, the LSN (log sequence number) of the previous oepration for that transaction is given.

LOG CONTENTS

LSN	Xact	Action	PrevLSN
1	T1	P1 update A B	-
2	T2	P3 update 1 2	-
3	T3	P4 update 4 8	-
4	T1	P2 update DD FF	1
5	T3	P4 update 8 12	3
6	T4	P5 update X1 X3	-
7	T1	commit	4
8	T5	P1 update B C	-
9	T3	commit	5
10	T2	P2 update FF HH	2
11	T4	abort	6
12	T4	CLR: undo 6	-
13	T5	P5 update X1 Y	8
14	T5	P4 update 12 20	13

DATA PAGE CONTENTS

Page	Content	LSN of last recorded change
P1	B	1
P2	HH	10
P3	1	0
P4	8	3
P5	X1	12

Answer. When we analyze this log, we find that transactions T1,T3 were committing and the rest should be aborted.

REDO step. We will only REDO the committed transactions (T1,T3). We go through each step and analyze whether REDO is needed.

LSN	Xact	Log Action	Recovery Action
1	T1	P1 update A B	P1 has LSN=1, no REDO is needed
3	T3	P4 update 4 8	P4 has LSN=3, no REDO is needed
4	T1	P2 update DD FF	P2 has LSN=10, no REDO is needed
5	T3	P4 update 8 12	P4 has LSN=3, REDO is needed
12	T4	CLR: undo 6	P5 has LSN=12, no REDO is needed

Now, we will abort T2,T5 together with T4, and undo their actions in reverse. Through analysis, we find the last actions of T2,T4, and T5:

T2: 10, T4: -, T5: 14. Since there is no prior action of T4 remains, we will remove it.

LSN	Xact	Log Action	Recovery Action
14	T5	P4 update 12 20	P4 has LSN=3, no undo is necessary
13	T5	P5 update X1 Y	P5 has LSN=12, no undo is necessary
10	T2	P2 update FF HH	P2 has LSN=10, UNDO (P2=FF)
8	T5	P1 update B C	P1 has LSN=1, no UNDO is necessary
2	T2	P3 update 1 2	P3 has LSN=0, no UNDO is necessary

Question 3. You are given the queries provided answers to Hw#5 and the jeopardy database from Hw #6 already created for you in your personal DB in the class server. You must find one index that reduces the cost of one of the queries in Hw#5. Document the improvement as described below. (Note that Query 9 is not applicable because the hw#6 dataset did not include the states table.)

To optimize a query, first you will look at its estimated cost before you create an index. To do this, you can simply add the word `explain` to the beginning of the query. For example:

```
jeopardy=> explain select fullname from contestants where  shortname='Gilbert';
                                      QUERY PLAN
-------------------------------------------------------------------------------------
 Index Scan using contestants_pkey on contestants  (cost=0.29..755.07 rows=4 width=13)
   Index Cond: ((shortname)::text = 'Gilbert'::text)
(2 rows)

jeopardy=> explain select fullname from contestants where  description like '%5-day%';
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on contestants  (cost=0.00..953.76 rows=2 width=13)
   Filter: ((description)::text ~~ '%5-day%'::text)
(2 rows)

jeopardy=> explain select fullname from contestants where  description like '%5-day%' order
                              QUERY PLAN
---------------------------------------------------------------------
 Sort  (cost=953.77..953.78 rows=2 width=13)
   Sort Key: fullname
   ->  Seq Scan on contestants  (cost=0.00..953.76 rows=2 width=13)
         Filter: ((description)::text ~~ '%5-day%'::text)
(4 rows)
```

In each query plan, you get two estimated costs (e.g. `0.29 and 755.07`). The first one is the time to the first answer (which is non-zero given the unavoidable cost of scanning the index first) and the second one is the cost of getting all the answers. In the second query, the initial cost is zero because you can start to produce tuples as soon as you start scanning the relation (assuming you find some matching tuples). You can see that the cost of the third query is the same for both first and all results because sort is a blocking query, you cannot return any results until the sort is complete. Then, you can return all the results.

To document your answer, simply show the query plan before creating the index, index creation command and the query plan after creating the index. Then, write down your savings, by computing `OLD-COST-NEW-COST`. This is the improvement we will look at.

Even if your answer was faster than mine, please use of one my answers and simply find one index that improves the running time and is used in the query plan.

Who will have the best improvement? Technically we only need one. But if you want to show us more than one, feel free.

Hint. Find the most costly queries. Do not run them, simply run `EXPLAIN`. These are the ones you can make a difference on. Look at conditions that have an equality and which attributes are involved in these.

Answer. I will choose to improve the performance of the most costly queries. Query 7:

```
  Cost before indexing:

CTE Scan on fivetimer f (cost=36357.52..36357.84 rows=1 width=226)
   Filter: (winnings = $4)
   CTE fivetimer
     -> GroupAggregate (cost=36356.92..36357.20 rows=14 width=21)
          Group Key: c.fullname
          Filter: (count(DISTINCT c.gameid) = 5)
          -> Sort  (cost=36356.92..36356.95 rows=14 width=21)
               Sort Key: c.fullname
               -> Nested Loop (cost=0.57..36356.65 rows=14 width=21)
                    -> Index Scan using scores_pkey on scores s (cost=0.29..36264.44
                       rows=14 width=14)
                         Index Cond: ((round)::text = 'Final Score'::text)
                         Filter: (score = (SubPlan 1))
                         SubPlan 1
                           -> Aggregate (cost=12.85..12.86 rows=1 width=4)
                                -> Index Scan using scores_pkey on scores s2
                                   (cost=0.29..12.85 rows=3 width=4)
                                     Index Cond: ((gameid = s.gameid) AND
                                         ((round)::text = 'Final Score'::text))
                    -> Index Scan using contestants_pkey on contestants c
                       (cost=0.28..6.59 rows=1 width=23)
                         Index Cond: ((gameid = s.gameid) AND ((shortname)::text =
                             (s.shortname)::text))
   InitPlan 3 (returns $4)
     -> Aggregate (cost=0.32..0.33 rows=1 width=8)
          -> CTE Scan on fivetimer (cost=0.00..0.28 rows=14 width=8)
(21 rows)


hw6=> create index abc on scores(round,gameid,score);
CREATE INDEX
```

```
CTE Scan on fivetimer f (cost=15800.29..15800.60 rows=1 width=226)
  Filter: (winnings = $5)
  CTE fivetimer
    -> GroupAggregate (cost=15799.68..15799.96 rows=14 width=21)
         Group Key: c.fullname
         Filter: (count(DISTINCT c.gameid) = 5)
         -> Sort (cost=15799.68..15799.72 rows=14 width=21)
              Sort Key: c.fullname
              -> Nested Loop (cost=73.36..15799.42 rows=14 width=21)
                   -> Bitmap Heap Scan on scores s (cost=73.08..15707.21 rows=14
                       width=14)
                        Recheck Cond: ((round)::text = 'Final Score'::text)
                        Filter: (score = (SubPlan 2))
                        -> Bitmap Index Scan on abc (cost=0.00..73.07 rows=2772
                            width=0)
                             Index Cond: ((round)::text = 'Final Score'::text)
                        SubPlan 2
                          -> Result (cost=5.58..5.59 rows=1 width=4)
                               InitPlan 1 (returns $1)
                                 -> Limit (cost=0.29..5.58 rows=1 width=4)
                                      -> Index Only Scan Backward using abc on scores
                                          s2 (cost=0.29..16.17 rows=3 width=4)
                                           Index Cond: ((round = 'Final Score'::text)
                                               AND (gameid = s.gameid) AND (score IS
                                               NOT NULL))
                   -> Index Scan using contestants_pkey on contestants c
                       (cost=0.28..6.59 rows=1 width=23)
                        Index Cond: ((gameid = s.gameid) AND ((shortname)::text =
                            (s.shortname)::text))
  InitPlan 4 (returns $5)
    -> Aggregate (cost=0.32..0.33 rows=1 width=8)
         -> CTE Scan on fivetimer (cost=0.00..0.28 rows=14 width=8)
(25 rows)
```

Savings: 36357-15800.

Note that this index also helps greatly in Query 10 as well, reducing the cost from 36470 to 15913.

Query 2: Minor improvements with index on scores(round).

Query 5: Minor improvements with index on contestants(fullname).