John Rivera

Homework 8

Database Systems

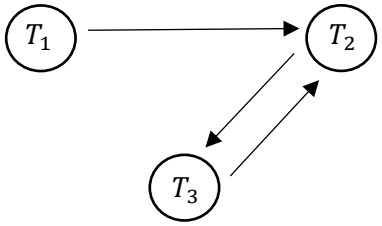
1.

S1 : r1(X) r1(Y) w2(Y) w2(Z) r3(Z) w3(K) r2(K) w2(L) w1(X)

a) Potential Conflicts:

$$r_1(Y), w_2(Y)$$
$$w_2(Z), r_3(Z)$$
$$w_3(K), r_2(K)$$

Conflict Graph:



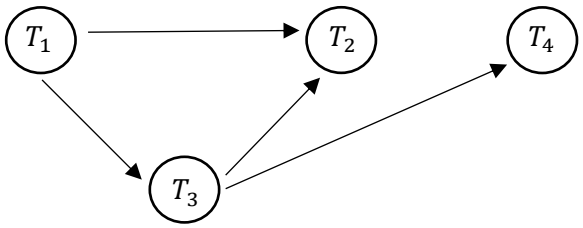
b) There is a cycle in the conflict graph, which means that these operations cannot occur in the same order as a serial schedule and S1 is therefore not serializable.

S2 : r1(X) w2(X) w1(Y) r3(Y) w3(Z) r2(Z) r3(W) w4(W) w2(Z) r4(W)

a) Potential Conflicts:

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

Conflict Graph:



b) There is no cycle in the conflict graph, which means that there exists an equivalent serializable schedule:

$$T_1, T_3, T_2, T_4$$

2.

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

ANALYSIS Phase:

Transaction Table (TT):

transaction	Last LSN	
T1	7	<committed>
T2	10	
T3	9	<committed>
T4	12	<abort>
T5	14	

DPT:

Page	LSN
P1	1
P3	2
P4	3
P2	4
P5	6

REDO Phase:

Only consider LSN's with committed transactions that update or CLR

So test these LSN's: 1, 3, 4, 5

Test LSN 1:

In DPT: P1, DPT_LSN = 1

LSN = 1 ≮ 1 = DPT_LSN

pageLSN = 1 = 1 = LSN

So do nothing

Test LSN 3:

In DPT: P4, DPT_LSN = 3

LSN = 3 ≮ 3 = DPT_LSN

pageLSN = 3 = 3 = LSN

So do nothing

Test LSN 4:

In DPT: P2, DPT_LSN = 4

LSN = 4 ≮ 4 = DPT_LSN

pageLSN = 10 > 4 = LSN

So do nothing

Test LSN 5:

In DPT: P4, DPT_LSN = 3

LSN = 5 ≮ 3 = DPT_LSN

pageLSN = 3 < 5 = LSN

REDO LSN 5

UNDO Phase:

Aborting T2, T4, T5

TO_UNDO = {10, 12, 14}

Write: UNDO 14 log record

P4 is already in memory but with pageLSN = 3

So no need to change the data page content as this update was never written to disk

prevLSN = 13

TO_UNDO = {10, 12, 13}

Write: UNDO 13 log record

P5 is already in memory but with pageLSN = 12

So no need to change the data page content as this update was never written to disk

prevLSN = 8

TO_UNDO = {10, 12, 8}

Write: UNDO 12 log record

prevLSN is NULL

write an end record for T4

TO_UNDO = {10, 8}

Write: UNDO 10 log record

P2 is already in memory with pageLSN = 10

So UNDO P2 contents and change it to FF

prevLSN = 2

TO_UNDO = {2, 8}

Write: UNDO 8 log record

P1 is already in memory with pageLSN = 1 < 8

So no need to change anything

prevLSN = NULL

write an end record for T5

TO_UNDO = {2}

Write: UNDO 2 log record

P3 is already in memory with pageLSN = 0 < 2

So no need to do anything

prevLSN = NULL

write an end record for T2

TO_UNDO = {}

Recovery Complete

3.

Query 10 plan before index:

QUERY PLAN
GroupAggregate (cost=36470.69..36470.71 rows=1 width=218)
Group Key: w.fullname
Filter: (count(DISTINCT w.gameid) >= 2)
CTE won
-> Nested Loop (cost=0.57..36356.65 rows=14 width=23)
-> Index Scan using scores_pkey on scores s_1 (cost=0.29..36264.44 rows=14 width=10)
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_1 (cost=0.29..12.85 rows=3 width=4)
Index Cond: ((gameid = s_1.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_1.gameid) AND ((shortname)::text = (s_1.shortname)::text))
-> Sort (cost=114.04..114.05 rows=1 width=222)
Sort Key: w.fullname
-> Nested Loop (cost=0.86..114.03 rows=1 width=222)
Join Filter: (((s3.shortname)::text <> (s2.shortname)::text) AND (w.gameid = s3.gameid))
-> Nested Loop (cost=0.57..113.31 rows=1 width=246)
Join Filter: (w.gameid = s2.gameid)
-> Nested Loop (cost=0.29..112.59 rows=1 width=236)
-> CTE Scan on won w (cost=0.00..0.28 rows=14 width=440)
-> Index Scan using scores_pkey on scores s (cost=0.29..8.02 rows=1 width=14)
Index Cond: ((gameid = w.gameid) AND ((shortname)::text = (w.shortname)::text) AND ((round)::text = '3'::text))
-> Index Scan using scores_pkey on scores s2 (cost=0.29..0.71 rows=1 width=14)
Index Cond: ((gameid = s.gameid) AND ((round)::text = '3'::text))
Filter: (((shortname)::text <> (s.shortname)::text) AND (score > s.score))
-> Index Scan using scores_pkey on scores s3 (cost=0.29..0.71 rows=1 width=14)
Index Cond: ((gameid = s.gameid) AND ((round)::text = '3'::text))
Filter: (((shortname)::text <> (s.shortname)::text) AND (score > s.score))

Created index:

```
create index scores_i1 on scores(round);
```

Query Plan after creating index:

QUERY PLAN
GroupAggregate (cost=36060.29..36060.31 rows=1 width=218)
Group Key: w.fullname
Filter: (count(DISTINCT w.gameid) >= 2)
CTE won
-> Nested Loop (cost=61.36..35946.25 rows=14 width=23)
-> Bitmap Heap Scan on scores s_1 (cost=61.08..35854.05 rows=14 width=10)
Recheck Cond: ((round)::text = 'Final Score'::text)
Filter: (score = (SubPlan 1))
-> Bitmap Index Scan on scores_i1 (cost=0.00..61.07 rows=2772 width=0)
Index Cond: ((round)::text = 'Final Score'::text)
SubPlan 1
-> Aggregate (cost=12.85..12.86 rows=1 width=4)
-> Index Scan using scores_pkey on scores s2_1 (cost=0.29..12.85 rows=3 width=4)
Index Cond: ((gameid = s_1.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_1.gameid) AND ((shortname)::text = (s_1.shortname)::text))
-> Sort (cost=114.04..114.05 rows=1 width=222)
Sort Key: w.fullname
-> Nested Loop (cost=0.86..114.03 rows=1 width=222)
Join Filter: (((s3.shortname)::text <> (s2.shortname)::text) AND (w.gameid = s3.gameid))
-> Nested Loop (cost=0.57..113.31 rows=1 width=246)
Join Filter: (w.gameid = s2.gameid)
-> Nested Loop (cost=0.29..112.59 rows=1 width=236)
-> CTE Scan on won w (cost=0.00..0.28 rows=14 width=440)
-> Index Scan using scores_pkey on scores s (cost=0.29..8.02 rows=1 width=14)
Index Cond: ((gameid = w.gameid) AND ((shortname)::text = (w.shortname)::text) AND ((round)::text = '3'::text))
-> Index Scan using scores_pkey on scores s2 (cost=0.29..0.71 rows=1 width=14)
Index Cond: ((gameid = s.gameid) AND ((round)::text = '3'::text))
Filter: (((shortname)::text <> (s.shortname)::text) AND (score > s.score))
-> Index Scan using scores_pkey on scores s3 (cost=0.29..0.71 rows=1 width=14)
Index Cond: ((gameid = s.gameid) AND ((round)::text = '3'::text))
Filter: (((shortname)::text <> (s.shortname)::text) AND (score > s.score))

So savings are:

36470.71 – 36060.31 = 410.4