Written Assignment 2: Summer-2(2023)

Priyanka Bojja: 700739528

1. Considering the join operation between relation r and s ($r \bowtie_\theta s$), θ is $r.A = s.B$ with
- Relation r contains 20,000 tuples and has 10 tuples per block.
- Relation s Contains 2,000 tuples and has 10 tuples per block.
- there are 17 buffer blocks available in memory(M).

(1.1) Using Block Nested Loop Join:

$$Cost = \left(\left\lceil \frac{b_r}{M-1} \right\rceil * b_s\right) + b_r$$

$$b_r = \frac{20,000}{10} = 2000$$

$$b_s = \frac{2000}{10} = 200$$

$$Cost = \left(\left\lceil \frac{2000}{17-1} \right\rceil * 200\right) + 2000$$

$$Cost = (125 * 200) + 2000$$

$$Cost = 27,000 \text{ block transfers}$$

(1.2) Using Merge Join:

$$B_s = b_r * \left(2\left\lceil \log_{M-1}\left(\frac{b_r}{M}\right)\right\rceil + 2\right) + b_s * \left(2\left\lceil \log_{M-1}\left(\frac{b_s}{M}\right)\right\rceil + 2\right)$$

$$= 2000 * \left(2\left\lceil \log_{17-1}\left(\frac{2000}{17}\right)\right\rceil + 2\right) + 200 * \left(2\left\lceil \log_{17-1}\left(\frac{200}{17}\right)\right\rceil + 2\right)$$

$= 2000 * (2\lceil 1.719 \rceil + 2) + 200 * (2\lceil 0.889 \rceil + 2)$

$= 2000 * (2(2)+2) + 200(2(1)+2)$

$= 2000 * 6 + 200 * 4 = 12,000 + 800 = 12,800$

Cost $= B_s + b_r + b_s$

$= 12,800 + 2000 + 200$

$= 15,000$ Block transfers

(1.3) Hash join ($r \bowtie s$): Non recursive partition:

Cost $= 3(b_r + b_s)$

$= 3(2000 + 200)$

$= 6,600$ block transfers

(1.4) Hash Join ($r \bowtie s$): Recursive partition:

Cost $= 2(b_r + b_s)\lceil \log_{M-1}(b_s) - 1 \rceil + b_r + b_s$

Where $(b_s \leq b_r)$

$= 2(2000+200)\lceil \log_{16}(200) - 1 \rceil + 2000 + 200$

$= 4,400\lceil 0.9 \rceil + 2000 + 200$

$= 6,600$ Block transfers

(1.5) If there is the infinity of memory, then the total minimum cost will be 6,600 Block transfers i.e; using hash join (No-recursive) because it is independent of memory for the cost calculation.

(2) SQL Query to find the instructor's name who teaches 'Advanced Database' in Fall Semester, 2021

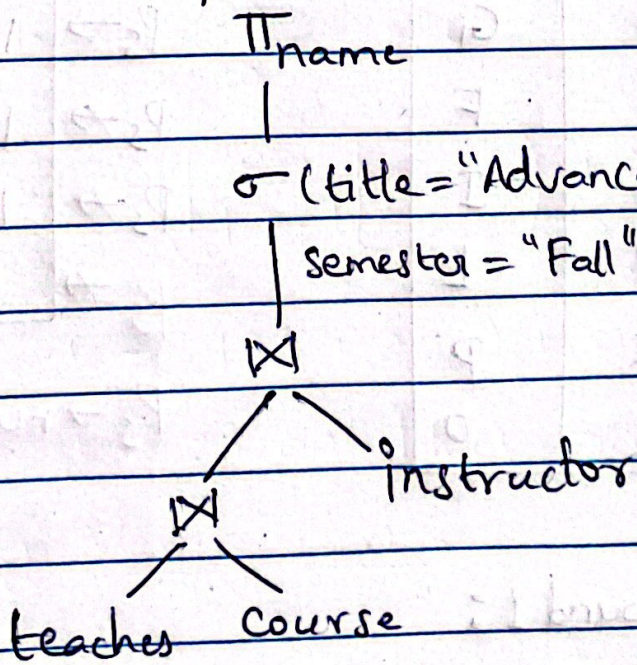
Query: SQL command:

```
SELECT name FROM instructor, teaches, course
WHERE teaches.course_id = course.course_id AND
instructor.id = teaches.id AND title = "Advanced
Database" AND semester = "Fall" AND year = 2021;
```

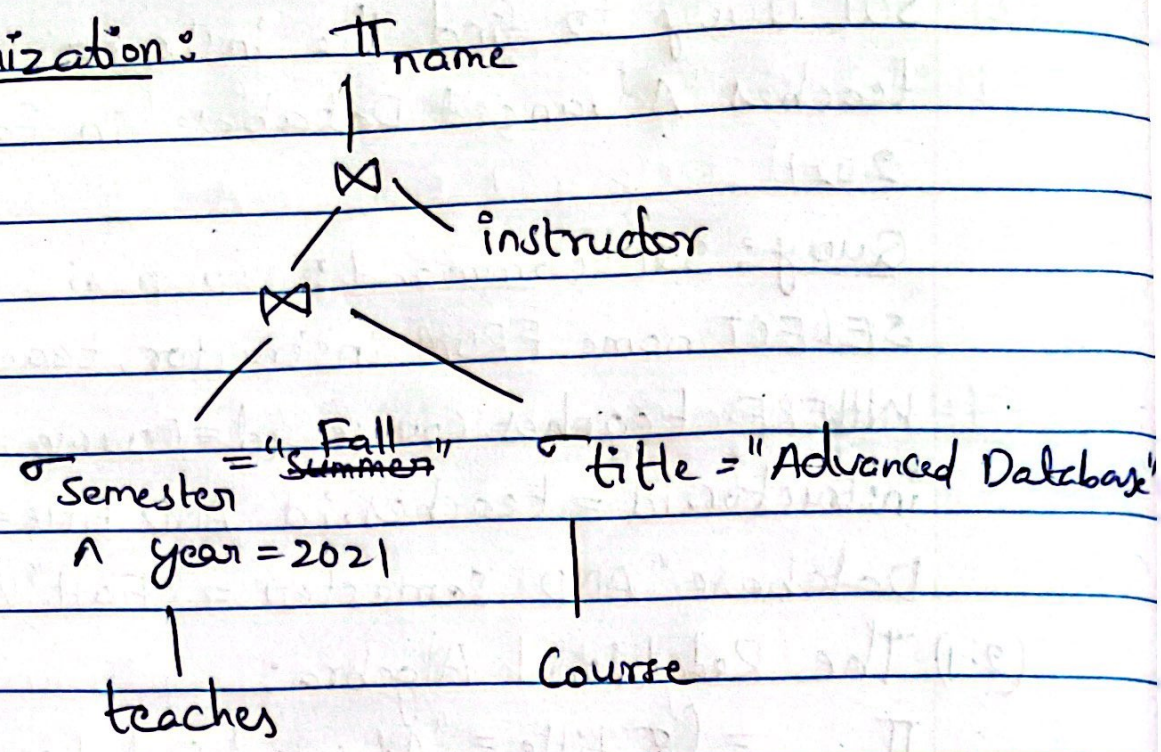
(2.1) The Relational Algebra:

$$\Pi_{name} = (\sigma \text{ title} = \text{"Advanced Database" AND semester} = \text{"Fall" AND year} = 2021 (\text{Instructor} \bowtie (\text{teaches} \bowtie \text{course})))$$

(2.2) Equivalent Expression:



optimization:



③ The Merge join with the samples relation R and S

	A_1	A_2	A_3
P_r	10	A	L
P_r	10	B	G
P_r	11	C	F
P_r	12	A	I
P_r	14	M	L
P_r	14	K	P
P_r	16	L	O

$P_r \rightarrow$ null

R

	A_1	A_4
P_s	10	30
P_s	12	40
P_s	12	50
P_s	15	20
P_s	16	10

S

$P_s \rightarrow$ null

Initially Round 1:

Where $P_r = (10, A, L)$ $P_s = (10, 30)$

$t_s = (10, 30)$ $S_s = \{(10, 30)\}$

updated $P_s = (12, 40)$

done: false $t_s' = (12, 40)$

Round #	t_s	S_s	t_r
1	(12, 40)	{(10, 30)}	(11, C, F)
2	(15, 20)	{(12, 40) (12, 50)}	(14, M, L)
3	(16, 10)	{(15, 20)}	(16, L, 0)
4	null	{(16, 10)}	null

updated done : true

$t_r = (10, A, L) \bowtie t_s(10, 30) \Rightarrow$

updated $P_r = (10, B, G)$

$t_r = (10, B, G) \bowtie t_s(10, 30) \Rightarrow$

updated $P_r = (11, C, F)$

$t_r = (11, C, F)$

Result

A_1	A_2	A_3	A_4
10	A	L	30
10	B	G	30
12	A	I	40
12	A	I	50
16	L	O	10

Round 2: $P_r = (11, C, F)$ $P_s = (12, 40)$

$t_s = (12, 40)$

$S_s = \{(12, 40)\}$

$t_s' = (12, 50)$

t_s in S_s add $t_s \bowtie t_r$ to result

$t_r = (12, A, I)$

updated $P_r = (12, A, I)$

$S_s = S_s \cup t_s' = \{(12, 40), (12, 50)\}$, $P_s = (12, 50)$

updated $P_r = (14, M, L)$, $t_r = (14, M, L)$, $P_s = (15, 20$

Round 3: $P_r = (14, M, L)$, $P_s = (15, 20)$, $t_s = (15, 20)$, $S_s = \{(15, 20)\}$

$t_s' = (16, 10)$

updated $P_r = (14, K, P)$

again updated $P_r = (16, L, 0)$, $t_r = (16, L, 0)$

$t_s = (16, 10)$, $S_s = \{(16, 10)\}$

Round 4: $P_r = (16, L, 0)$, $P_s = (16, 10)$, $t_s = (16, 10)$

$S_s = \{(16, 10)\}$, $P_s \rightarrow$ null

$P_r \rightarrow$ null, $t_r \rightarrow$ null

(4)

S1

T_1	T_2	T_3
Read(z)		
write(y)		
	Read(x)	
	write(x)	
	Read(z)	
		Read(x)
		write(x)
	write(z)	
	commit	
Read(x)		
commit		
		Read(y)
		commit

S2

T_1	T_2	T_3
Read(z)		
write(y)		
Read(x)		
commit		
	Read(x)	
	write(x)	
	Read(z)	
	write(z)	
	commit	
		Read(x)
		write(x)
		Read(y)
		commit

4.1 In schedule S1 the conflicts are as follows:

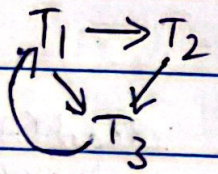
T_1 Read(z) with T_2 write(z)

T_1 write(y) with T_3 Read(y)

T_2 Read(x) with T_3 write(x)

T_2 write(x) with T_3 Read(x)

T_3 write(x) with T_1 Read(x)



In schedule S2 the conflicts are as follows:

T_1 Read(z) with T_2 write(z)

T_1 ~~Read~~ write(y) with T_3 Read(y)

T_1 read(x) with T_2 write(x)

T_1 read(x) with T_3 write(x)

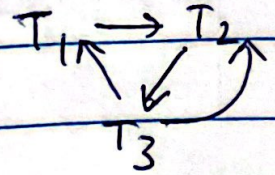
T_2 read(x) with T_3 write(x)

T_2 write(x) with T_3 read(x)

T_3 read(x) with T_2 write(x)

T_3 write(x) with T_2 Read(x)

T_3 read(y) with T_1 write(y)



By comparing the set of conflicts, we can see that S1 and S2 have different sets of conflict. Therefore S1 is not conflict equivalent to S2.

4.2 View equivalent where

		S1 X	Y	Z
I	IR	T_2	T_3	T_1
II	P/C	$T_2 \to T_3$ $T_3 \to T_1$	$T_1 \to T_3$	$T_1 \to T_2$
III	LUP	T_3	T_1	T_2

S2	X	Y	Z
I IR	T_1	T_3	T_1
II P/C	T_2/T_3	T_1/T_3	–
III LUP	T_3	T_1	T_2

Therefore S1 and S2 do not produce the same values for all data items. So, S1 is not view equivalent to S2.

5. Schedule S3

T_1	T_2
Lock-X(A)	
Read(A)	
A = A+100	
	Lock-S(B)
	Read(B)
	Lock-S(A)
write(A)	
unlock(A)	
Lock-X(B)	
	Read(A)
	unlock(B)
	commit
	unlock(A)
	Display(A+B)
Read(B)	
B = B−100	
write(B)	
commit	
unlock(B)	

5.1) In schedule S3, we can identify the following dependencies and locks:

→ T_1 acquires a lock(X) on A before reading and updating.

→ T_2 acquires a Lock(S) on B before reading.

→ T_2 acquires a shared lock on A After releasing the lock on B.

The deadlock doesnot occur in schedule S3 Although there is a dependency between T_1 and T_2, the locking mechanism allows for compatibility between two transactions. Therefore no deadlock in S3.

5.2) The schedule S3 already guarantees Serializability with T_1 followed by T_2. T_1 holds an exclusive lock on A and perform its operations before releasing the lock. T_2 acquires shared lock on B and A and commits. No modification is necessary to guarantee Serializability. T_1 followed by T_2.