CSE 411 / L-10 / 30.09.2024

## Quize-1

### Block nested loop

* Worst case:

Block Transfer = number of block of first relation *
number of block of second relation
+ number of block of first relation

$= b_R * b_S + b_R$

Seek $= b_R + b_S$

Best Case:

Block Transfer = $b_R + b_S$

Seek = 2

### Evaluation of Expression

i. Materialization:

- generate results of an expression whose inputs are relations or already computed.

- Materialize: store on disk.

ii. Pipelining:

- pass ~~one~~ on tuples to parent operations even as an operation is being executed.

Query-1:

```
SELECT  Name
FROM    Instructor  INNER JOIN  Department
        ON  Instructor.dept-name = Department.dept-name
WHERE       Building = "Watson"
```

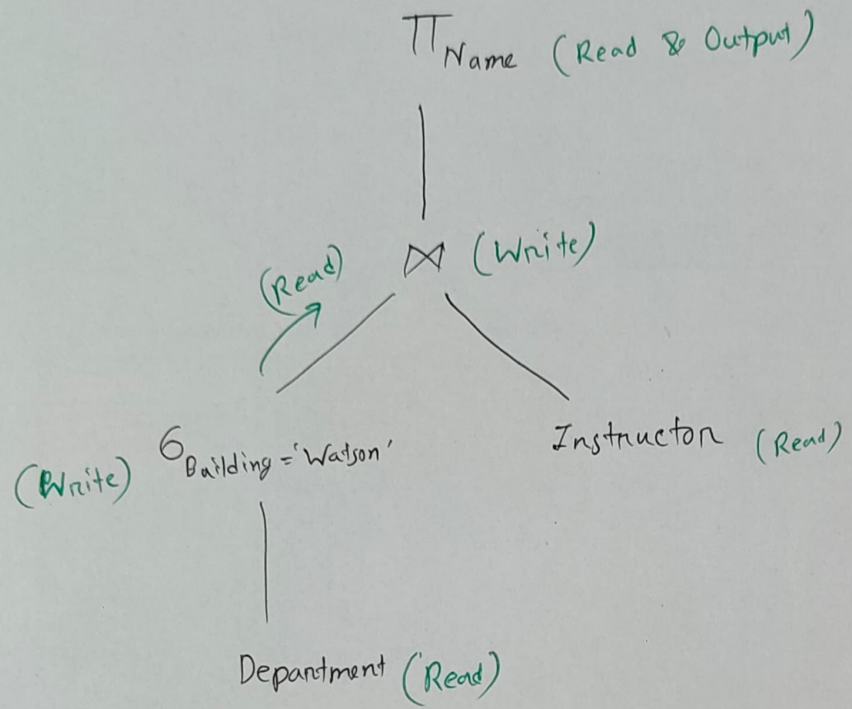
⇒ Algebra:

$$\Pi_{Name}\left(\left(\sigma_{Building = Watson}(Department)\right) \bowtie Instructor\right)$$

⇒ Execution will start from the inner parenthesis. But, we will construct the execution tree from outer parenthesis.

As in our classical computer, we call function in another function, like nested, then we get the result on return call one by one. Execution tree is like that, it first goes to the leaf, & once leaf is reached, it start to return the result to parent and finally we get the output from the root.

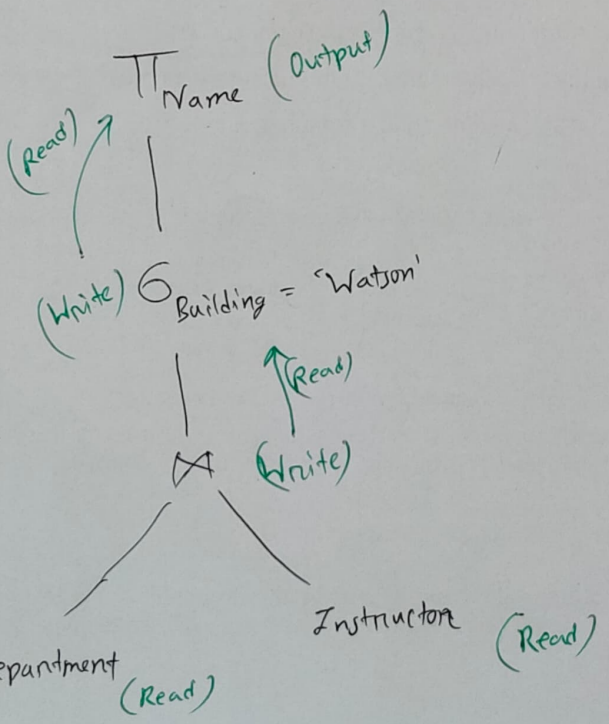
Execution tree:




Algebra:

$$\Pi_{Name}\left(\sigma_{Building='Watson'}\left(Department \bowtie Instructor\right)\right)$$

Execution Tree:



L-11 / 02.10.2024

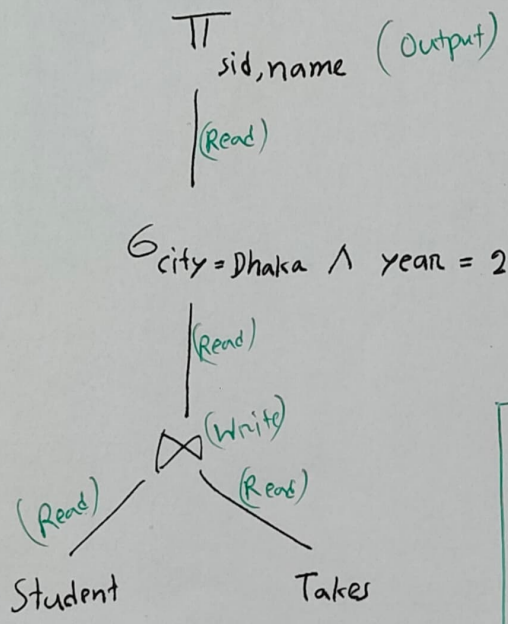
| Midterm |
| --- |
| 23.10.2024 |
| upto Transaction |
| Lecture Slide- 0,1,2 |

From the sample Question:

(a)

Algebra:

$$\Pi_{sid,\,name}\left(\sigma_{city = 'Dhaka' \wedge year = 2023}\left(Student \bowtie Take\right)\right)$$

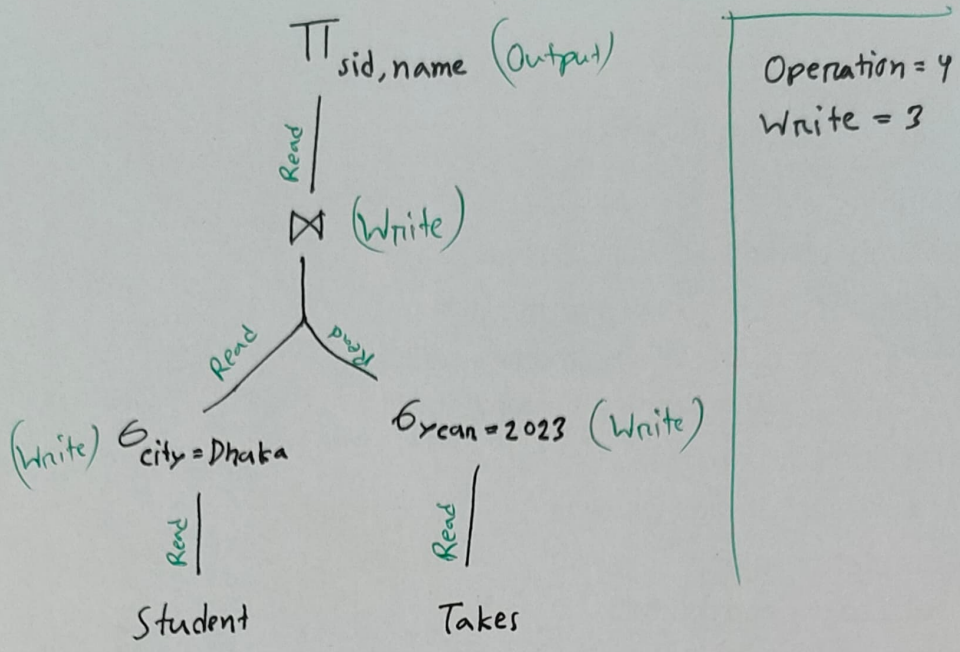



Operation = 3
Write = 2

still it will take more time, because of large join operation.

⇒ Optimized:

$$\Pi_{sid,\,name}\left(\sigma_{city = Dhaka}(Student) \bowtie \sigma_{year = 2023}(Takes)\right)$$




- For read & write, cost is high for materialization.

Overall cost = $\Sigma$ operation cost + $\Sigma$ write cost

- Problem-13:

Overall cost = $\Sigma$ operation cost + $\Sigma$ write cost

$$= (1S + 20B + 10S + 100B + 1S + 50B) + (1S + 10B + 1S + 50B)$$

$$= 14S + 230B$$

Ans

- Materialization method is so costly for read/write. There is another method to reduce this read write waiting time.

Double buffering:

Double buffering:

- two buffer for each operation
- when 1 buffer is full, it start to write on disk and in the mean time other buffer store the operation result.

## Pipelining

⇒ No result will be store on disk. Once one tupple found that match with the condition, it will send the data to the next operation directly.

⇒ Pipelining is not possible for sorting or hash-join.

Pipelining method can be executed in two ways:

(i) Demand driven (lazy evaluation) ⇒ Bottom level wait for the request from Top level. (Pull model)

(ii) Producer driven ⇒ Bottom level generate output and push it to top level for further operation. (Push model)

- eager
- maintain buffer - puts & remove.

# Transaction Manager

## Transaction:

- is a unit of program execution that accesses and possibly updates various data items.

## Maintain:

- from hardware failure, system crash
- concurrent execution of multiple transaction.

## ACID Properties:

### Atomicity:

- the system should ensure that, updates of a partially executed transaction are not reflected in the database.

### Consistency:

- when the transaction completes successfully the database must be consistent.
- Like: sum of balance of all account must be same as before.

### Isolation:

- when multiple transaction are executed concurrently, each transaction must be unware of other transaction
- one transaction can't effect others.

## Durability:

- after a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

L-12 / 07.10.2024 /

### Transaction States

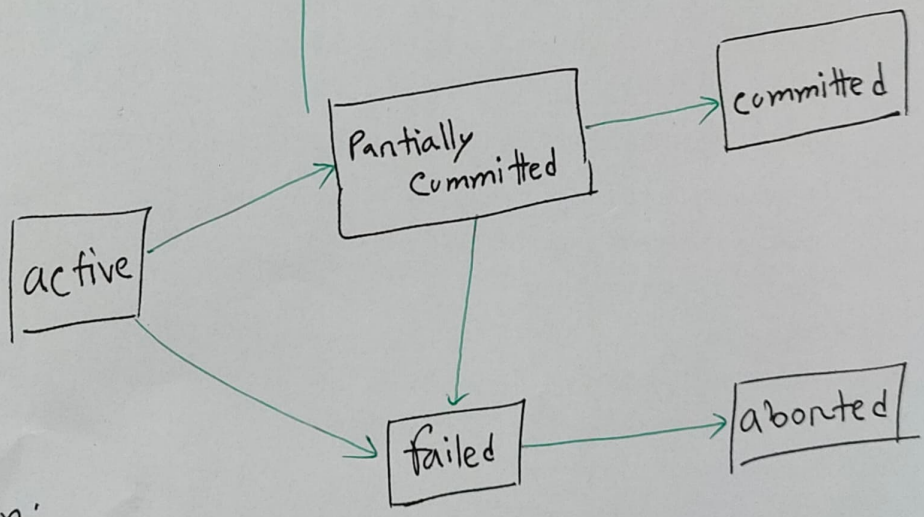
- Active
- Partially Committed
- Failed
- Aborted → Restart / kill
- Committed

T:

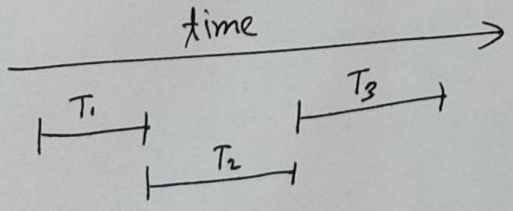
START
READ (A)
A = A - 10,000
WRITE (A)
COMMIT

Failed State

→ Partially Committed




### Serial Execution:

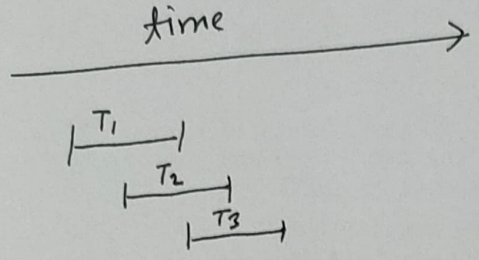



- Throughput high/low?
  ⇒ Low.
- No integrity problem.

⇒ Integrity = Yes

- Throughput ⇒ number of transaction per second.

## Concurrent Transaction:




- integrity problem may exist.
- maintained by "concurrency control schemes" - mechanism to achieve isolation.

* Throughput high/low?
  ⇒ High.

* Throughput also means resource utilization. Where resource utilization is high/max/100%, there throughput is must be high.

## Advantage of concurrency:

- increased processor & disk utilization.
- reduces average response time.

## Schedule: → Serial, → Concurrent

- transaction sets must be consisted.
- preserve the order in which the transaction appear.

*
- Schedule-1 ⇒ serial ⇒ database consistent.
- schedule-2 ⇒ serial ⇒ database consistent
- schedule-3 ⇒ concurrent ⇒ consistent
- schedule-4 ⇒ concurrent ⇒ not consistent.

## Serializability:

- conflict
- view

- concurrent schedule can be serializable, if the schedule is equivalent to a serial schedule.

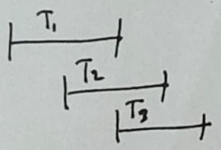
⇒

consistency exist ⇒ serializable

" not exist ⇒ not serializable.

Let, there are three transaction: $T_1$, $T_2$, $T_3$

concurrent schedule:




Serial transaction possibility:

| | | |
|---|---|---|
| $T_1$ | $T_2$ | $T_3$ |
| $T_1$ | $T_3$ | $T_2$ |
| $T_2$ | $T_1$ | $T_3$ |
| $T_2$ | $T_3$ | $T_1$ |
| $T_3$ | $T_1$ | $T_2$ |
| $T_3$ | $T_2$ | $T_1$ |

⇒ if concurrent schedule is equivalent to any of the serial schedule, then it is serializable.

⇒ And if it is serializable, then it must be consistent.

Schedule only depends on read/write operation.

Conflicting instruction:

- occure when read/write operation requested in the same object/account.

Between two transaction one the same account:
↳ two or more than.

- if both are read operation, then no conflict.
- in all other combination, there must be a conflict.
- write operation must be done independently.

Example : slide - 20, 21

L-13 / 09.10.2024

Example:

| $T_1$ | $T_2$ |
|---|---|
| READ (P) | |
| | WRITE (Q) |
| READ (Q) | |
| | WRITE (R) |
| WRITE (Q) | |
| WRITE (R) | |

conflict

⇒

| $T_1$ | $T_2$ |
|---|---|
| | WRITE (Q) |
| | WRITE (R) |
| READ (P) | |
| READ (Q) | |
| WRITE (Q) | |
| WRITE (R) | |

⇒ Serializable

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| READ (A) | | |
| | READ (A) | |
| READ (B) | | |
| | READ (B) | |
| | | WRITE (D) |

⇒ Possible to serialize in $T_1, T_2, T_3$

⇒ Serializable

all read operation - no conflict

write operation but no other operation on D.

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| READ (P) | | |
| | WRITE (Q) | |
| | WRITE (R) | |
| READ (Q) | | |
| WRITE (Q) | | |
| | | READ (R) |
| WRITE (R) | | |

$\Rightarrow$ $T_2, T_3, T_1$

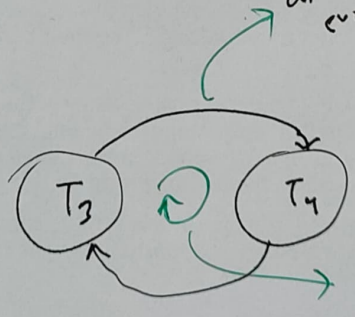
= Serializable

* Swaping method takes too much time and space. There is another algorith regarding the graph theory.

$\Rightarrow$ Precedence Graph

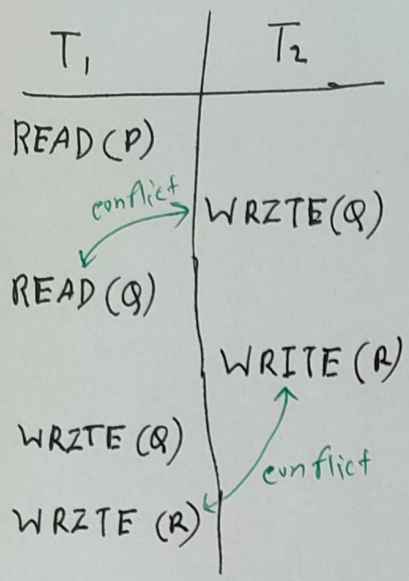
- vertices are the transaction

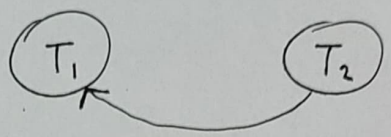
| $T_3$ | $T_4$ |
|---|---|
| READ(Q) | |
| | WRITE (Q) |
| WRITE(Q) | |




| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| READ(A) | | |
| | READ(A) | |
| READ(B) | | |
| | READ(B) | |
| | | WRITE(D) |

$\Rightarrow$ No conflict operation
- no graph
- serializable.




Same arch for two conflict because of $T_2$ is in up than $T_1$

⇒ No circle
- serializable

✳ If precedence graph is acyclic, then serializability order can be obtained by a topological sorting of the graph.

Midterm Syllabus upto this

## ✳ Concurrency Control

✳ Lock mechanism to control concurrent access:

two modes of lock:

(i) exclusive (X) ⇒ Read + Write permission

(ii) shared (S) ⇒ Only read permission

For concurrent lock requests

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

if both are 'S' mode, then allowed otherwise not allowed.