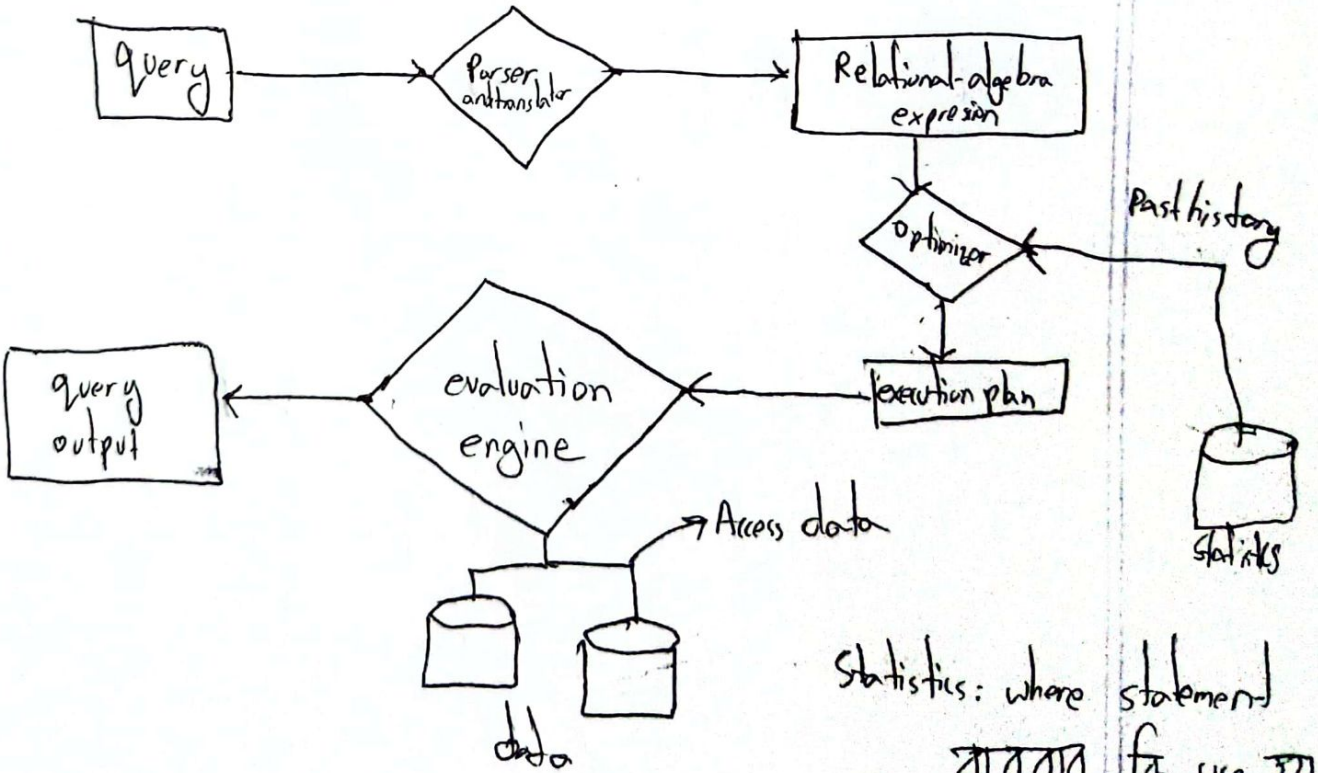
CPU time → time to parse query

elapsed time → Time to retrieve query and display query results

Query Optimization

Query processing algorithm



Statistics: where statement বাড়লে কি use হয়, tuple কি কি access হয়

Relational algebra expression :

$\sigma_{salary < 75000}(\Pi_{salary}(instructor))$ equivalent to

$\Pi_{salary}(\sigma_{salary < 75000}(instructor))$

[Select salar from instructor where salary < 7500]

Query optimization: Amongst all equivalent evaluation, choose the one with lower cost.

Query cost : Disk Cost + Access Cost + CPU processing cost

↓

Highest cost

Need to find average-seek-cost to find block
+ average block read cost
+ average block write cost

Transfer cost: $b * t_t + S * t_s$ t_t = time to transfer one bl
t_s = time t

If block is kept in buffer, performance better, but

As keep more blocks, needs more buffer space.

Selections: Linear search cost : $(b_r/2)$ block transfer + seek

Using indexes : h_i = height of B+ tree

primary key index : (equality on key):

Cost : $(h_i + 1) * (t_T + t_S)$

Join operations :

↳ Nested loop join

↳ Block nested loop join

↳ Indexed nested loop join

$r \bowtie_\theta s$

for each tuple t_r in r do begin

for each tuple ~~t_s~~ t_s in s do begin

test (t_r, t_s) if they satisfy, join

r is outer relation, s in inner.

Since all pairs need to be checked this is

Total cost

$n_r * b_s + b_r$ block transfers

↓ → Access all blocks of relations

for every tuple of relation r

$n_r + b_r$ total seek

Join of students and takes:

Number of records: Student: 5000 Take: 10000

Number of blocks: Student: 100 takes: 400

Student outer relation

Block transfers: ~~5000 * 10000 + 500~~

$5000 * 400 + 100 = 2000100$ block transfer

Seeks: $5000 + 100 = 5100$

But if take is outer:

Block transfers: $10000 * 100 + 400 = 1000400$ much less

Block nesting: Instead of matching tuple, match every block

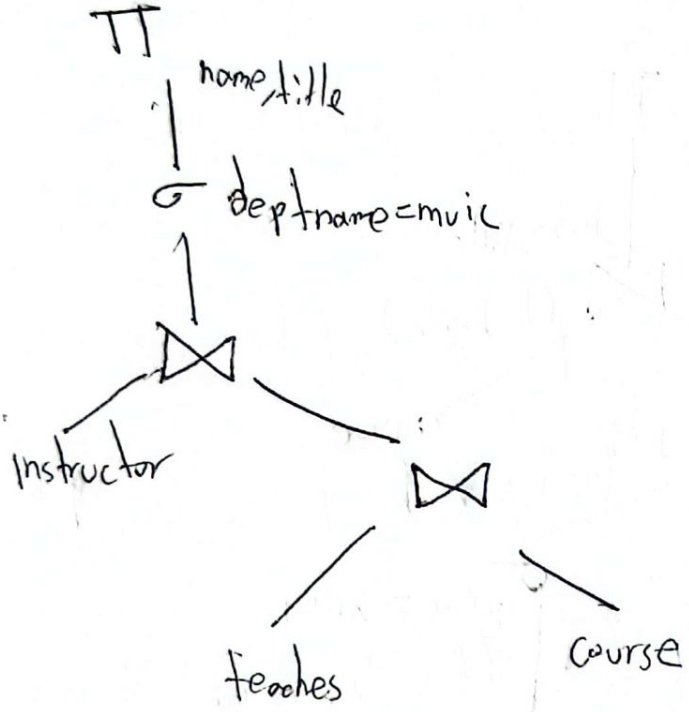
↳ Cost will come down by a factor of $M-2$ where M = memory unit in size blocks

$$\text{Cost} = \lceil b_r/(M-2) \rceil * b_s + b_r \text{ block transfer}$$

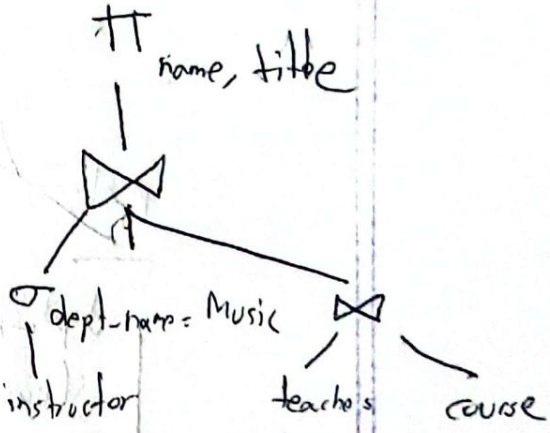
$$2\lceil b_r/(M-2) \rceil \text{ seeks}$$

Evaluating query :

course (courseID, title, deptname)
instructor (ID, name, deptname)
teaches (ID, courseID, secID, sem)



First join, then sele



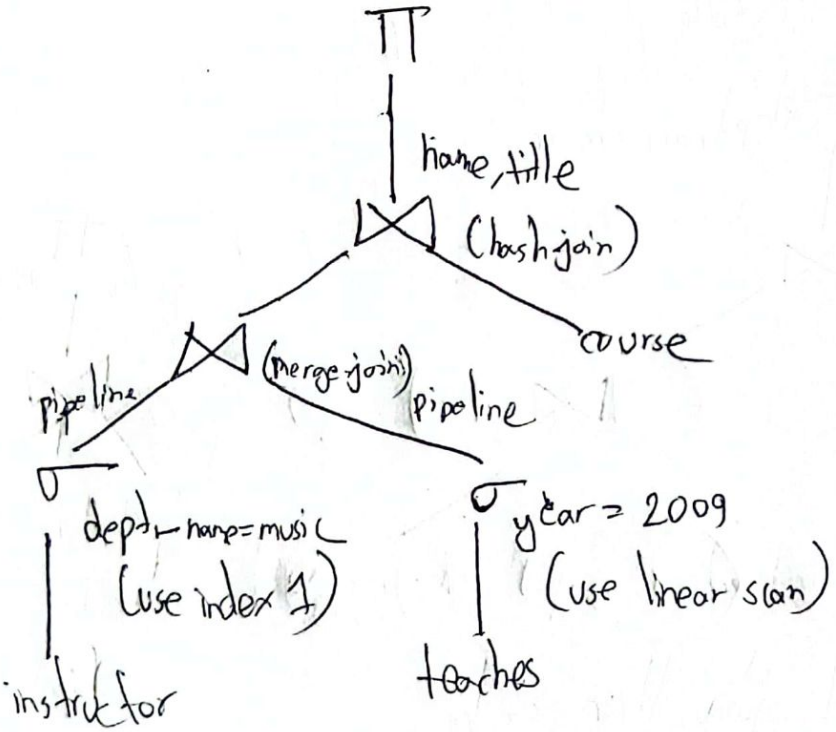
First selad then join

↓

These are equivalent

Second one is ~~better~~ better

$$\Pi_{name,\ title}\left(\sigma_{dept\_name\ =\ "Music"\ \wedge\ year\ =2009}\left(instructor \bowtie teaches \bowtie course\right)\right)$$



Query optimization:

→ Generate candidates using equivalence rules

→ Alternative query plans

→ Choose the cheapest plan using estimated cost

Cost: number of tuples number of distinct values

Transformation of relational expressions:

Equivalence rule ⟶ Generate two expression that generates the same tuples.

↳ Order of tuples are irrelevant.

Rules:

Conjunctive selection

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

This is commutative

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

But cost will be different

In sequence of projections

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E)))\ldots)$$

$$= \Pi_{L_1}(E)$$

Other projections can be ignored

Important

$$\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$$

$$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$$

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

Cost can be different

* $(F_1 \bowtie F_2) \bowtie F_3 = F_1 \bowtie (F_2 \bowtie F_3)$

* $(F_1 \bowtie_{\theta_1} F_2) \bowtie_{\theta_2 \wedge \theta_3} F_3$

$= F1 \bowtie_{\theta_1 \wedge \theta_3} (F2 \bowtie_{\theta_2} F3)$

Provided θ_2 present in F2 and F3

* When all attributes in θ_0 involve only one F_1 being joine

$\sigma_{\theta_0} (F_1 \bowtie_\theta F_2) = (\sigma_{\theta_0}(F_1)) \bowtie_\theta F_2$

* $\sigma_{\theta_1 \wedge \theta_2} (F_1 \bowtie_\theta F_2) = (\sigma_{\theta_1}(F_1)) \bowtie_\theta (\sigma_{\theta_2}(F_2))$

8 (a)

$\Pi_{L1 \cup L2} (F_1 \bowtie_\theta F_2) = (\Pi_{L_1}(F_1)) \bowtie_\theta (\Pi_{L_2}(F_2))$

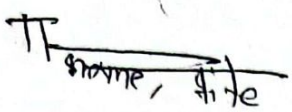
$\Pi_{L1 \cup L2} (F1 \bowtie_\theta F_2) = \Pi_{L1 \cup L2} ((\Pi_{L1 \cup L3}(F_1)) \bowtie_\theta (\Pi_{L3 \cup L4}(F_2)))$

Query: Find the names of all instructors in the Music dept who have taught a course in 2009, along with the titles of the courses that they taught.

$$\Pi_{name,\ title}\left(\sigma_{dept\_name\ =\ "Music"\ \wedge\ year\ =\ 2009}\left(instructor \bowtie \left(teaches \bowtie \Pi_{course\_id,\ title}(course)\right)\right)\right)$$

$$\Pi_{name,\ title}\left(\sigma_{dept\_name\ =\ "Music"\ \wedge\ year\ =\ 2009}\left(\left(instructor \bowtie teaches\right) \bowtie \Pi_{course\_id,\ title}(course)\right)\right)$$

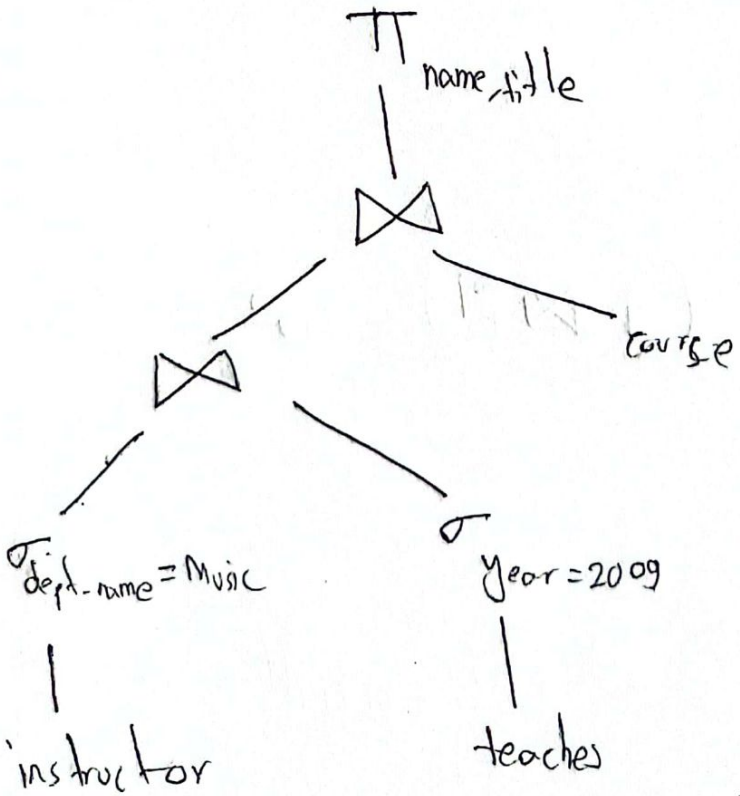
course(courseID, title, dept-name, credits)
instructor(ID, name, dept-name, salary)
teaches(ID, course id, secid, semester, year)



$\sigma_{dept\_name = "Music"}(instructor) \bowtie \sigma_{year = 2009}(teaches)$

Do this, it is better

* instructor ⋈ teaches selects all tuples, so ~~select first then join~~ this is bad!!



↳ Equivalent and better

Push projections :

$$\Pi_{name, title}\left(\left(\sigma_{dept\_name = \text{"Music"}}(instructor) \bowtie teaches\right) \bowtie \Pi_{course\_id, title}(course)\right)$$

Practice

key না হয় হলে।

1টা block এ কয়টা

Total tuple → n_r

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

$V(A, r)$ → r relation A attribute কয়টা

$$sl(A, r) = \left\lceil \frac{n_r}{V(A, r)} \right\rceil$$

n_r → Total tuple

$V(A, r)$ → A attribute r relation এ কয়টা

↓

Selection Cardinality

↓

গড়ে কয়টা attribute আছে block এ

Binary এখন binary search করে per block এ একটা attribute খুঁজতে কতক্ষণ লাগবে?

Ans: $\log b_r + \left\lceil \frac{sl(A,r)}{f_r} \right\rceil - 1$

$f_{ac} = 20$

$V(b.name, a_c) = 50$

$V(bala, ac) = 500$

$n_{ac} = \cancel{100000}\ 10000$

select branch_name="perridge" from account

$b_{ac} = \left\lceil \frac{n_{ac}}{f_{ac}} \right\rceil = 500$ $\quad \frac{St(A,r)}{f_r}$

$\log(500) + \frac{10000}{\cancel{100000}50} - 1$

$= \log(500) + \frac{10000}{20} - 1$

$= \log(500) + 10 - 1$

Equility on key : B+ Tree height + 1

Equility on primary index : B+ Tree height + $\left\lceil \frac{3+}{fr} \right\rceil$

Equility on key ~~ना रहेगा~~

~~Next~~ Fanout = 20 Node=50

treeheight (min) = 3

Equility on key = 3+1 = 4

Equility on primary index = $3 + \left\lceil \frac{10000}{\frac{50}{20}} \right\rceil$

$= 3 + 10 = 13$

DBMS Query

Select from webpage where author = Joh

σ → select

(σ index lookup (Webpage) author = "John"

⋈ index-join occurs url = url

⋈ main-memory-hash-join σ index-lookup wid=wid language = 'french' (Word)

X

⟕ → Left outer join → Left side সবসময়ই হবে Right Null হবে

X → যা মিলবে Null

⋈ → Inner join

Table join

Select author from webpage where author = "fmh" and language = "French";

$T(Webpage) = V(Occurs, url) = 10^9$

$T(Occurs) = 10^{12}$

$T(word) = 10^6 = V(Occurs, wid)$

$v(webpage, author) = 10^7$

$v(word, language) = 100$

Block size = 10

$Sc(Webpage, Author) = \left\lceil \frac{T(webpage)}{V(Webpage, Author)} \right\rceil$

→ Distinct average in total tuple

→ Total Data

$= \frac{10^9}{10^7} = 10^2$

→ Number of author in per block average

$Sc(Occur, url) = \frac{T(Occur)}{}$

$Sc(Occur, URL) = \left\lceil \frac{T(Occur)}{V(Occurs, url)} \right\rceil$

$= \frac{10^{12}}{10^9} = 10^3$

Join $Sc(Webpage, Author) \bowtie Sc(Occur, URL) = 10^5$

Main-memory তে block access লাগে না। So, ~~block access~~

Primary index হলে 10 দিয়ে ভাগ??

3rd ১০ জন্য কোনো কিছু নাই

~~T(word) = 10~~

$$Sc(Word, language) = \left\lceil \frac{T(word)}{V(Word, language)} \right\rceil$$

$$= \frac{10^6}{100} = 10^4$$

Webpage. url = primary

webpage. auth = secondary

occurs. url = secondary

occurs. wid = primary

word. wid = primary

word. language = secondary index

Total =

~~$10^5 + 0 + 10^4$~~ ~~$10^2 + 10$~~

Total = $10^2 + 10^5 + 10^4$

10^2 ↓ first lookup

10^5 ↓ Join

10^4 → 4th lookup

Primary কে secondary দিয়ে replace.

a   Url

$$\text{(i)}\ \frac{10^2}{10} + \underset{\text{(ii)}}{\frac{10^5}{10}} + \frac{10^4}{10} + 0$$

$$\underset{\text{(i)}}{\frac{10^2}{10}} + \underset{\text{(ii)}}{10^2 \times \frac{10^3}{10}} + \underset{\text{(iv)}}{\frac{10^4}{10}} + \underset{\text{(iii)}}{0}$$

$$= 10 + 10^4 + 10^3 + 0$$

=

Left side এ যা থাক দেখা লাগবে না

Right এ primary থাকলে আমাকে বারবার

বারে যাওয়া লাগবে so 10 দিয়ে ভাগ

∞

Transaction

A transaction is a unit of program execution that accesses possibly updates various data items.

To ~~pre~~ preserve integrity of database, a system must ensure:

Atomicity: Either all operations of the transaction are properly reflected in the database or none are.

Consistency: Execution of a transaction in isolation preserves consistency of database

Isolation: Transaction can occur concurrently although each transaction must be unaware of the other executing transaction.

Durability: After a transaction completes successfully the changes it made to the database persist.

States:

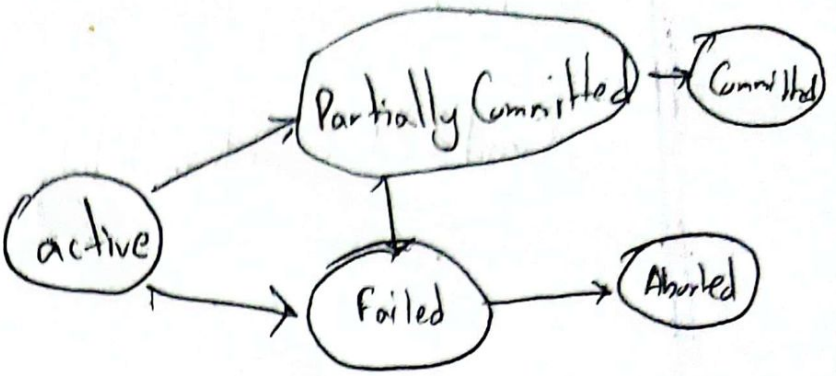
Active

Partially committed

Failed

Aborted

Committed



Concurrent executions : Multiple transaction are allowed to run concurr
in system

~~Schedule~~

Schedules : Sequence of instructions that specify the chronological order in which ~~concurrent~~ instructions in concurrent transactions are executed.

A transaction that fails to successfully complete its execution will have an abort instruction as last statement

Serializability:

Each transaction preserves database consistency

Conflict serializability

A	B
r	r
T	w
w	r
r	w
w	

If a schedule S can be transformed into a schedule S' by a series of non-conflicting instructions, we say S and S' is conflict equivalent.

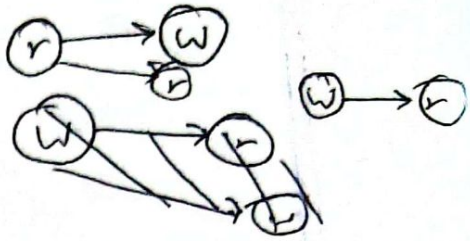
Schedule S is conflict serializable if it is conflict equivalent to serial schedule.

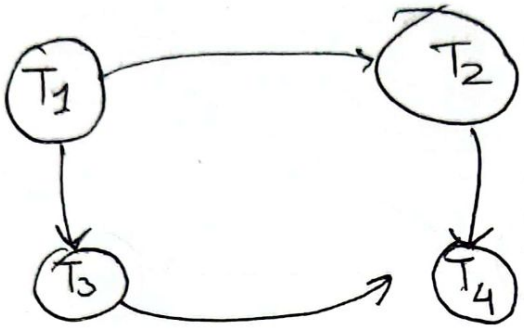
View Serializable

A schedule S is view serializable if its equivalent to a serial schedule.

S [illegible] তা আছে, S' এ তা থাকতে হবে

Testing serializability: Precedence graph





If precedence graph is acyclic, serializability order can be obtained by ~~top~~ topological sor

A	Pre	
1	1	TP
0	0	TN
0	1	FP
1	0	FN

Recovery System : Database চলতে চলতে crash করলে tran
-saction recover করতে সাহায্য,
stable system design → Copy
→ Disperse elements in HDD

Log based recovery
Shadow based recovery

Log based recovery → Deferred database modificat
↑ Postpone করা

যা করার : Log এ করব, পরে গিয়ে system এ add কর
So crash করলে Log এর data loss হয়

<T0 start>
<T0, A, 950>
<T0, B, 2050>

<T0 start>
<T0, A, 950>
<T0, B, 2050>
<T0 commit>
<T1 start>
<T1, C, 600>
(b)

<T0>.

Commit হওয়ার আগে crash করলে no need to

Atomicity: Minimum individual unit. সোটা কাজ টা ঘটবে, partial হতে পারে না
A give B. A−50, B+50 দুটোই হবে

Consistancy: দিন শেষে হিসাব মিলতে হবে,
A ও B এর মোট টাকা same.

Isolation: Parallel operation must occur in isolation. A-50 B+50 আলাদা operation

Durability: পৃথিবী ধ্বংস হলেও হিসাব থাকবে

Inconsistane state এ থাকলে back যায়.

redo<T0> if database is not done. R. Idipotent.
তাহলে redo করলে ও কোনো



I2 => I

$$\text{Confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{Sup}(A \cup B)}{\text{Sup } A}$$

$$I2 \Rightarrow \{I1, I5\} = \frac{2}{7}$$

Immediate DB modification :

Immediately DB modify হয়

So, log এ আগের value present value থাকে

<T0 start>
<T0, A, 1000, 950>
<T1, B, 2000, 2050>

<T0 start>
<T0, A, 1000, 950>
<T0, B, 200, 2050>
<T0 commit>

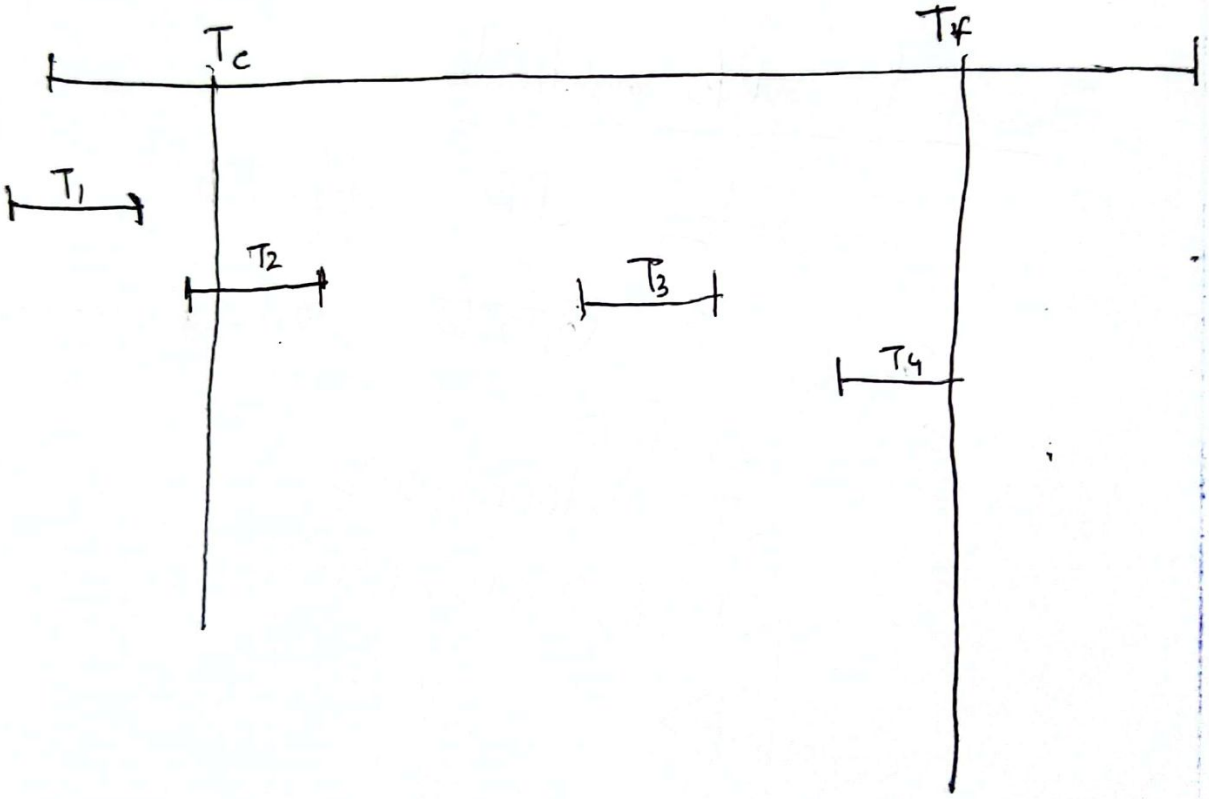
↓

Undo এবং redo

Example of Checkpoints.

Database recovery : Undo redo crash করলে তখন কোথা থেকে শুরু করব

Checkpoint: Flag. প্রত্যেক ফিল্ড বোঝানো যে সবকিছু ঠিক আছে



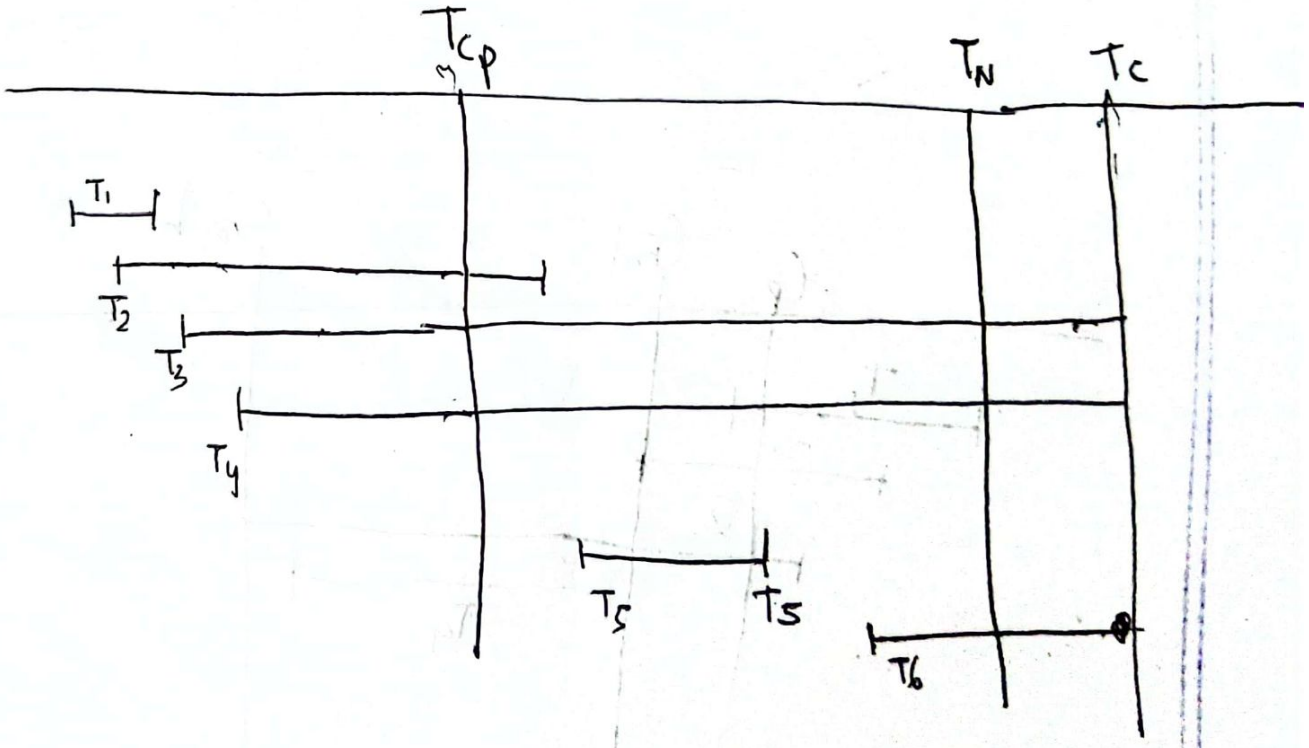
T_2 → redo

T_3 → Redo

T_4 → Undo করে redo (Commit হয় নাই)

<Ckpt (T_2, T_3, T_4)>

<End Ckpt>



~~T_2 → Redo T_3 → Undo and Redo T_4 → Und~~ and redo

T_1 → ~~फिरसे~~ नही T_2 → Redo

T_3 → Undo & ~~Rep~~ Redo

T_4 → "

~~T_5 → Redo~~

T_5 → Redo

T_6 → Undo

~~T_6 → Undo and redo~~

A → B B → 20

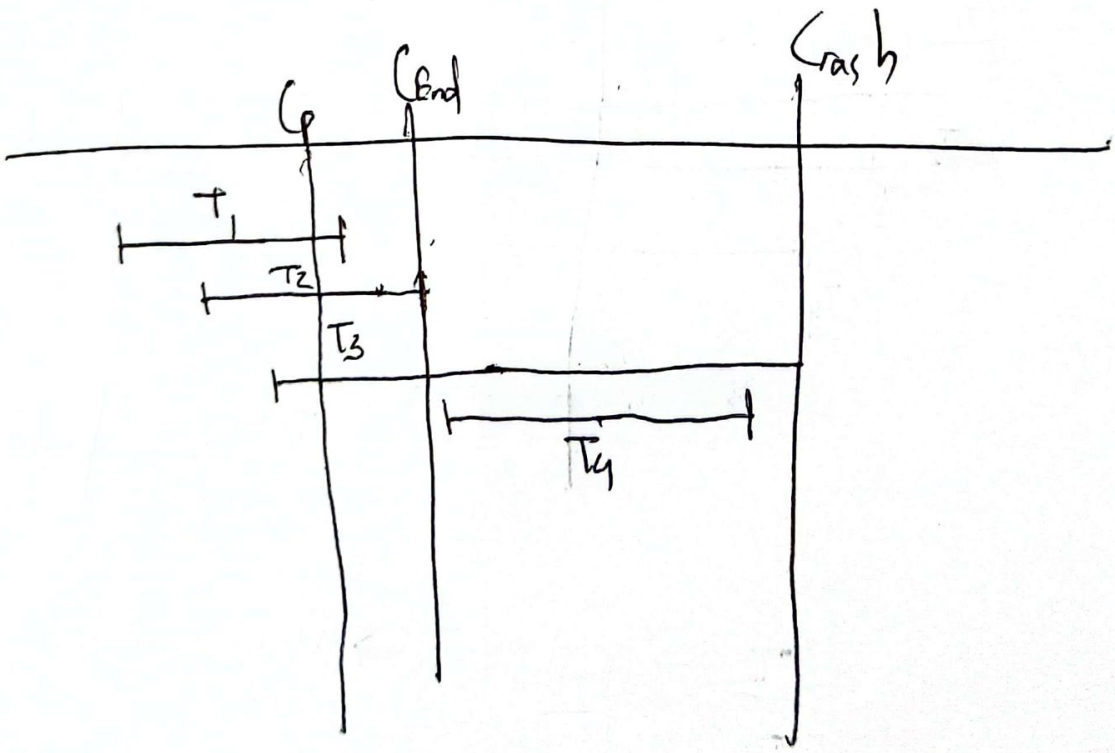
D → 2000

C → 200

(b) start T_1>

P

<



T_3 → undo and redo T_4 → redo

DBMS

Concurrency Control:
A database must be:
↳ Conflict Serializable
↳ Recoverable and possibly cascadeless

Goal → To develop concurrency protocols that will assure serializability
↳ Cascadeless database

Make transactions ACID

I → Isolation → Once a portion is at use others cannot use it → Bad for concurrency

So, Lock must be in such a way concurrency has to ensured.

Lock-based protocols: Two ~~and~~ modes

Exclusive mode: Two try to write, you do not have who ~~write~~ have written.

Shared mode: Anyone can read at any time

Transaction can be X-lock S-lock and unlocked using unlock(Q)

Lock-compatibility mode:

	S	X
S	true	false
X	false	false

Reading and writing together is incompatible and vice versa.

If a lock cannot be granted, the tx must wait until all locks are released.

Let A and B be two Tx on T1 and T2

* Tx T1 transfers $50 from B to A.
* Tx T2 displays total mount of money in A and B, A+B

A = $100 B = $200

T1:

lock-X(B)
read(B)
B := B - 50;
write(B)
unlock(B);
lock-X(A);
A := A+50;
write(A)
unlock(A);

T2:

lock-S(A)
read(A);
unlock(A);
lock-S(B);
read(B);
unlock(B)
display(A+B);

If ~~T1 and T2~~ T2 run after T1 then Concurrency is maintained.

Another concurrency

T1	T2	Concurrency control manager
lock-X(B)		grant-X(B, T1)
read(B)		
B := B-50		
write(B)		
unlock(B)		

T1	T2	Concurrency control manager
	lock-S(A)	
		grant-S(A, T_2)
read(B)		
B := B - 50		
write(B)		
unlock(B)		
	lock-S(A)	
		grant-S(A, T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B, T_2)
	read(B)	
	unlock(B)	
	display(A+B)	
lock-X(A)		
		grand-X(A, T_1)
A := A + 50		
write(A)		
unlock(A)		

This is concurrent. But here Total is 50$ is less.

Hence, this is not proper concurrency

If unlocking is pushed to the end.

T3:

lock-X(B)

:

lock-X(A)

:

unlock(B)
unlock(A)

T4:

lock-S(A)
read(A)

Here, T4 will correctly show sum

T3	T4
lock-X(B)	
:	
	lock-S(A)
	read(A)
	lock-S(B) → B is locked
lock-X(A) →	T4 holds shared

This is "deadlock".

When deadlock occurs, system must rollback transactions.

Locking protocol → Conflict serializability must be ensured.

Two phase locking protocol :

Phase - 1 : Grow Phase

↳ Tx may obtain locks

→ Tx may not release locks

Phase 2 : Shrinking phase

↳ Tx may release locks

→ Tx may not obtain locks

lock conversions in two phase locking

first phase: Can obtain lock-S
" " lock-X
upgrade lock-S to lock(X) [upgrade]

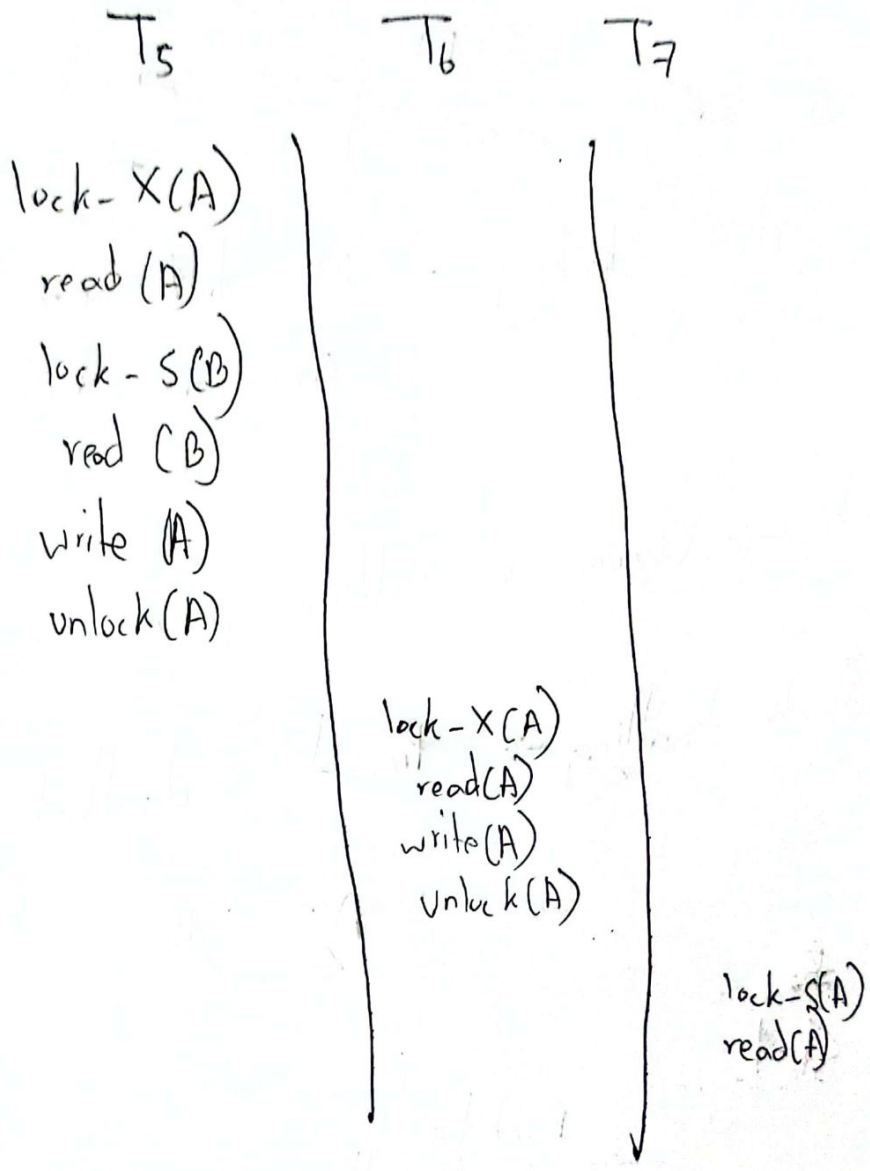
Second phase: Release lock-S
Release lock-X
downgrade lock-X to lock-S

Starvation → lock ধরে আর ছাড়ে না, → starvation,
কোনো tx lock পায়ই না
Starvation occurs if concurrency is badly designed.

↳ Can cause repeated ~~roll~~ deadlocks and roll-back

Deadlocks can occur in any system, they are necessary evil.

Posibility of cascading roll Back



Here After failure of T5 to read A

Prevent cascading roll-back, strict two phase locking → All exclusive locks are held till commit or abort

OR even stricter ⇐ rigorous two phase locking → All locks are held till commit or abort

Lock → Occurs on different process

Deadlock handling → Prevent deadlock.

Strategies

→ Each tx Lock all data items before execution

→ Partial ordering of all data items, lock items in that order

Problem: Brings down concurrency
Delay transactions

Schemes → wait-die schen → non preemptive

Older transaction should ~~never~~ wait for younger, younger transactions never wait for older ones, they are rolled back.

A tx may die several times before acquiring lock

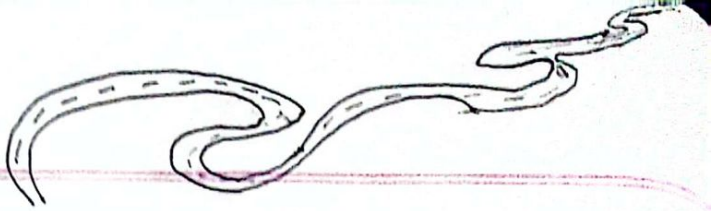
Wound - wait → preemptive

Older tx preempts younger tx

younger tx wait for older ones

Timeout based schemes

↳ Starvation possible



Recovery:

~~ACID~~ ACID

- Atomicity
- Consistency
- Isolation
- Durability

- Transaction failure:
  - Logical errors
  - System errors
- System crash
- Disk failure

Recovery algorithm:

1. During normal tx, collect info to recover from failures
2. Ensure atomicity, consistency and durability of the tx.

Failure during data transfer can result in inconsistent copies.

Block transfer can result in successful completion, partial failure and total failure

Protecting storage media from failure during transfer

↳ Write the information into two physical blocks, first followed by second

Copies of a block may differ. ~~To~~ during failure

To recover from failure:

1. Find inconsistent blocks

   ↳ If error in checksum, replace error

2. If no error, replace second by first.

Data Access :

Physical block → ~~Main m~~ disk block

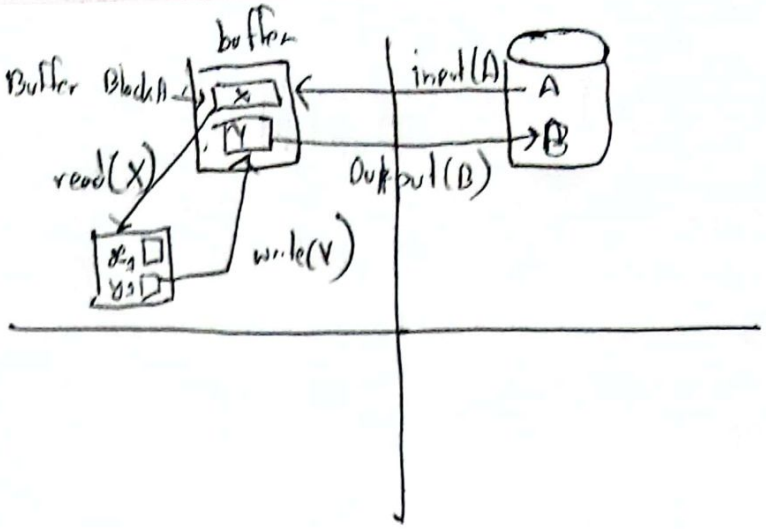
System buffer blocks → blocks temporarily residing in main memory

input(B) → physical block (B) to main memory

output(B) → transfers the buffer block B on to the disk

read (X) → assign ~~vari~~ the value of data item X to local variable x_i

write (X) → assign the value of local variable x_i to buffer block.



Log based recovery mechanism

A log is kept on stable storage

Database modification

Update log must be written before a database item is written

Immediate modification → allow updates to the buffer before tx commits

Deferred modification → performs updates only at the time of tx commit.

Log Write Output

<T_0 start>

<T_0, A, 1000, 950>

<T_0, B, 2000, 2050>

A = 950

B = 2050

<T_0 commit>

<T_1 start>

<T_1, C, 700, 600>

C = 600

Contains updated value of B

B_B, B_C → Update of file, C, happening before T_1 commit

<T_1 Commit>

B_A

Undo and redo operations

Undo → Revert the effect of an update

Redo → Doesn't revert to old.
Rewrite the new value

<T0 start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>

(A)

undo(T0): B restored to 2000 and A to 10000

<T0 start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0 commit>
<T1 start>
<T1, C, 700, 600>

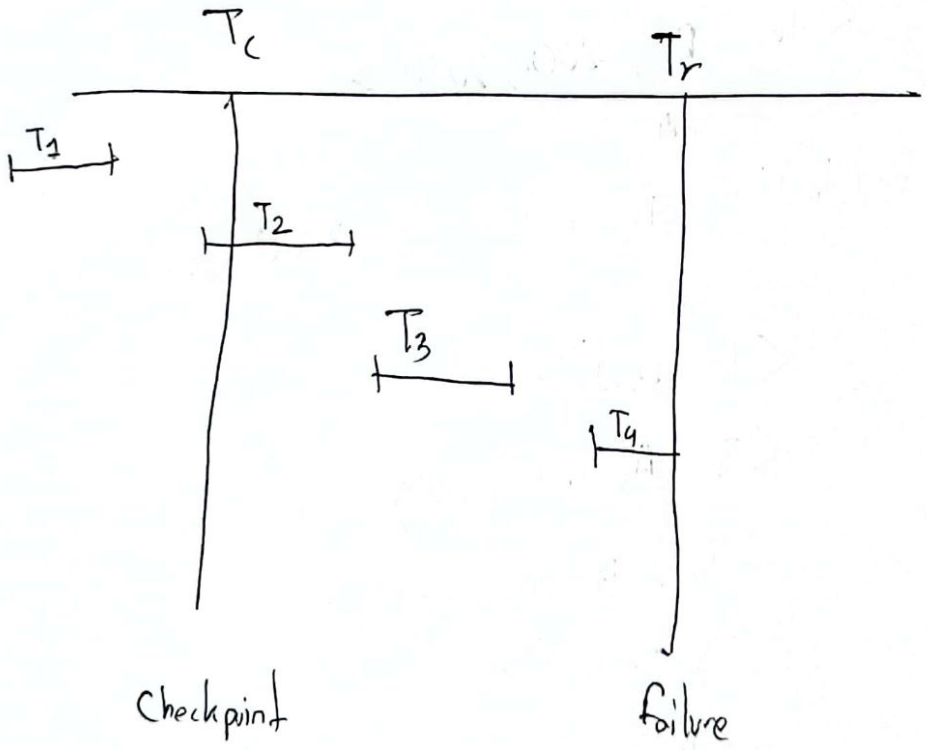
redo(T0) and undo(T1)

Commit 2Q 2PC to fix the tx → Do redo

Checkpointing:

Completed tx are fully secured after passing checkpoint

This time, we can just go back to the checkpoint and which tx were live during checkpointing



T_2 → redo

T_3 → redo

T_4 → Do not know final results. Undo

Concurrency control and recovery

<T0 start>

<T0, B, 2000, 2050>

<T1 start>

<checkpoint {T0, T1}>

<T1, C, 700, 600>

<T1 commit>

<T2 start>

<T2, A, 500, 400>

<T0, B, 2000>

<T0 abort>

Crash → <T2, A, 500>

<T2 abort>