Ch01 Introduction:-

* purpose of database systems: - reduce difficulty in accessing data {programming built} & storing data {data structures built}
  - Atomicity of updates
  - concurrent access by multiple users
  - security problems

* Schema & Instances
  ]
  ~~an entry in table~~.
  snapshot of table at some instance.

→ Data definition language DDL

* for defining schema of database.

```
create table instructor (
    ID      char(5)          (fixed length)
    name    varchar(20),
    dept_name varchar(20),
    Salary  numeric(8,2))
```

we have
- Schema
- Integrity constraints
  - primary key
- Authorization
  - who accesses what.

→ Data manipulation language DML (or query language

* Basically two types:

Procedural DML: user specifies what data & HOW to get it.

Declarative DML: user says what data but doesn't specify how...
]
just declare what you want.
NO involved code.

* Database schema: physical ; logical.

Database Design: physical, logical

Database Engine:

- Storage Manager
- Query Processor — To optimize our queries
- Transaction Management Component. — ensures database remains in a consistent state; even while power failures

* Architectures:-

- Centralized Database
- Client-server
- Parallel databases
- Distributed databases

ch2 : Intro to relational Model:-

relation ≡ Table.

* Each table consists of columns (Attributes) & rows (Tuples)

- special value "null" is member of every domain.

instance means, snapshot of tables all tuples.

→ Keys:-

* let K ⊆ R. K is superkey of R if values of K are sufficient to identify unique tuple in R.

{ID} & {ID, dept_name} are both superkeys for Instructor.

* Superkey K is candidate key, if its minimal.

- one of the candidate keys is choosen as primary key.

* foreign key constraint:- value in one relation should appear in another relation.

: referencing relation

: referenced relation

→ Relational Algebra:-

* This is not turing-machine equivalent. 6 basic operations.

- A procedural language, consisting of set of operations that take one or two relations as input, & produce new relation as output

select : σ $\sigma_{dept\_name = "physics"}(instructor)$

project : Π $\Pi_{A_1, A_2, \ldots A_k}(r)$

union : $\cup$

set difference : $-$

cartesian product : $\times$ $r \bowtie_\theta s = \sigma_\theta(r \times s)$ (JOIN)

rename : ρ

$\rho_x(E)$

$\rho_{x(A_1, A_2, \ldots A_k)}(E)$

* Assignment operator: temporary relational variable

$$Physics \leftarrow \sigma_{dept\_name = "Physics"}(Instructor)$$

Ch3 - Introduction to SQL:-

IBM developed SQL

→ Data definition Language:- (of SQL)

* SQL's DDL allows specification for
  - schema for each relation
  - value types for each attribute
  - integrity constraints
  - set of indices, to be maintained for each relation.
  - authorization for each relation.
  - physical storage structure of each relation.

1. Create tables:-

```
create table instructor (
    ID          varchar(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department);
```

end of SQL query.

2) Update to table schema:-

- Insert into instructor values ('10211', 'Smith', 'Physics', '66000');
- delete from students where CPI < 6;
- drop table instructor;
- Alter table instructor add
  weight int(3,2);
- Alter table demons drop name;

Basic query

```
Select a1, a2, ...
from r1, r2, ...
where
    r1.name = r2.name and
    ...
```

* SELECT clause. ✓

SELECT DISTINCT for distinct values.

SELECT ALL $A_1, A_2, \ldots$ from ...

to allow duplicates

* can have arithematic.

> Select runs*0.1 / overs*0.1 as avg_runs

* WHERE clause. ✓

<, >, <=, =, <> ; and, or, not

not equal to

>WHERE cond1 and
cond2 and
⋮
condk

* Ordering tuples

```
SELECT ...
FROM ...
WHERE ...
ORDER BY name asc, rank
                    desc;
```

* RENAME using 'as'; for attributes and relations.

→ String operations

* we have two string matching operators:

(%) matches substrings

(_) matches single character.

where name LIKE '%dar%'

Note:-

1) BETWEEN comparision operator :-

where Salary between 9000 and 10000;

i.e. 90,00 ≤ Salary ≤ 10,000

→ Set operations on relations:-

( ) union ( ), ( ) intersect ( ), ( ) except ( ) } automatically eliminates duplicates.

use
union all
intersect all
except all
to retain duplicates.

* IS NULL is used to check if attribute value is null.

Eg: WHERE salary is null and
name is not null;

→ Aggregate Functions:-

* These operate on multiple values of a column.

> avg, min, max, sum, count (even more exist... rank(), dense_rank())

Eg: Select count(*)
From teaches;

Select count(distinct ID)
From instructors;

→ Group by: ** Awesome guy!

* Attributes in SELECT clause without aggregate function must appear in GROUP BY clause.

Eg: SELECT dept_name, ID, avg(salary)
FROM teaches
GROUP BY dept_name, ID

1) Having clause :-

```
select deptname, avg(salary) as salary_avg
from instructor
group by dept_name
having salary_avg > 10,000;
```

different from 'WHERE' clause

IMP

* predicates in WHERE clause are applied Before forming relation & those in HAVING clause are applied After forming relation.

2) Nested Subqueries :-

i) in select clause :-

Ai replaced by subquery that generates single value.

ii) in from clause :-

we write subquery in ( )s.

iii) in where clause :-

```
where salary in (....) and
      salary not in (...);
```

3) Set Membership :-

- where course_id in (select ...
  from ...
  where ...;)
- where course-id not in ( );

4) Set comparision :-

i) SOME clause:

where salary > some ( );
(sub query)

ii) ALL clause:

where salary > all (----);
(subquery)

5) Test for empty relations :-

* EXISTS clause returns true if argument is not empty.

exists r $\Leftrightarrow$ $r \neq \phi$

not exists r $\Leftrightarrow$ $r = \phi$

useful for queries like "students who have taken all courses of maths"!

* UNIQUE clause is a test for absence of duplicates.

where UNIQUE (...); is true; if (...) relation has no duplicates.

→ Subqueris in FROM clause :-

```
* Select dept_name, avg_salary
  FROM ( Select dept_name, avg(salary)
         FROM instructor
         GROUP BY dept_name)
       as dept_avg (dept_name, avg_salary)        renaming the subquery.
  WHERE avg_salary ≥ 40,000;
```

* WITH clause : * Good one

- provides a way of defining a temporary relation, visible only to query in which its written

```
WITH dept_total (dept_name, value) as
(
   ...
),                          usable immediately.
dept_avg (value) as
( select avg(value)         } also part
  from dept_total)          } of with clause.
SELECT * FROM dept_average;
```

→ Scalar Subquery:- Scalar huh...

* used where single value is expected. runtime_error if more than one returned

```
Eg: select dept_name,
        (select count(*) from instructor } Scalar subquery.
         where department.dept_name = instructor.dept_name)
         as num_instructors } renaming.
    from department;
```

* All we've done is ask, ask, ask. Time to modify the database.

1) Deletion of tuples:-

```
delete
from instructor
where salary < (select avg(salary) from instructor);
```

scalar subquery.

computed only once.

2) Insertion of tuples:

* insert into course
  values ('CS-347', 'Comp. Sci', 4);
* insert into course (credits, course_name, dept_name)
  values (4, 'CS-347', 'Comp Sci');
* we can insert a relation into a table.

```
INSERT INTO instructor
SELECT * FROM instructor2
WHERE Salary > 10,000
```

no values() keywords

completely evaluated before inserting

3) Update tuples:-

```
UPDATE instructors
SET Salary = Salary * 1.05
WHERE
    Salary < (select avg(salary) from instructors);
```

Q) Increase salaries of people whose salary over 100K by 3%, and all others by 5%.

A) we can write two SQL queries... but CASE clause is more appropiate.

* case statement for conditional update:-

(same question as before)

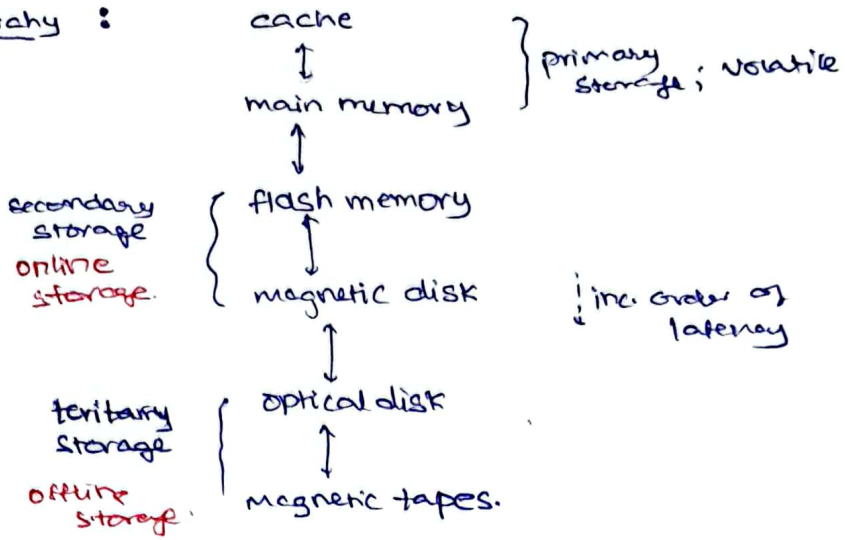
```
UPDATE instructor
  SET  salary = CASE
                WHEN salary <= 100,000 then salary*1.05
                ELSE salary*1.03
                END;
```

* instead of sum(credits); use

```
case
when  sum(credits) is not null then sum(credits)
else  0
end;      to set total-creds to 0, when its null.
```

12. Physical Storage Systems:-

* **Storage heirarchy :**



* **disk interface Speeds:**

SATA < SAS < NVMe (non-volatile memory express.)

6GBps 12GBps 24GBps

↳ works with PCIe connectors.

→ **Magnetic disks:-**

- **Accesstime:**

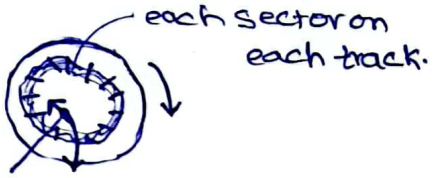
Seek time: reposition arm over correct track.

- Avg. time is $\frac{1}{2}^{th}$ of worst time

Rotational latency:

go to the correct sector.

- Avg. latency is $\frac{1}{2}^{th}$ of worst time.



→ **Performance measures:-**

1) **Disk blocks:-** logical unit for storage allocation and retrival.

- Smaller: more disk operations means
- larger : more space wasted. means

2) **Sq. Access pattern:-**

- Successive reqs. are for successive disk blocks.
- disk seek required only for 1st block.

3) **Random access pattern:-**

- sucessive requests are random.
- Each access needs a seek.

4) IOPS: I/O per seconds

5) MTTF:- avg. time; disk is expected to run continuously.

= 1,200,000 hours.. for new disks

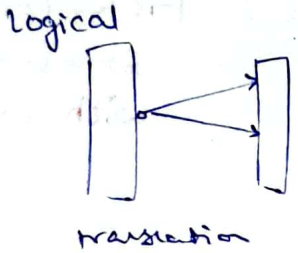
→ Flash storage:

* Not much difference b/w sq. and random read.

- pages can be written only once; must be erased to re-write.
- remappings logical page addr. to physical page addr. avoids waiting for erase

Flash translation table tracks mapping

> Also stored in label field of flash page. (But why!)



Logical address & valid bit stored with each physical page. quiz error!

→ RAID:-

redundant arrays of independent disks

> disk organization technique; to manage a large number of disks:

- high capacity & speed, using disks in parallel.
- high reliability by storing data redundantly.

Eg) mirroring/shadowing: duplicate every disk.

mean time to data loss; depends on MTTF & mean time to repair.

$$MTDL = \frac{(MTTF)^2}{2 \cdot MTTR}$$

* **Block Striping:-**

n disks; block i goes to i(mod n)+1. of a file. — not any redundancy.

Raid L0:

- non-redundant
- block-striping

[a] [b] [c] — used in high perf. when safety is not priority.

Raid L1:

- mirrored disks
- block striping.

[a] [b] [c] [a] [b] [c] — best write performance.

Parity blocks:

Store XOR of disks.

→ when new entry in the disk, recompute parity block.

- use old disk
- use new disk. (2 read + 2 write)
- use old parity.

(here only 2 writes) → Prefer for small size storages, with small random updates.

Raid L5: block-interleaved distributed parity.

parity blocks; stored across all N+1 disks; rather than single disk.

so N+1 parallelism in block read/write. (nice)

D_0	D_1	D_2	D_3
P_0	0	1	2
3	P_1	4	5
6	7	P_2	8

→ Prefer for large amounts of data storage, writes sequential.

Raid L6: stores 2 parities; to account for multiple disk failures.

* **optimization of disk block access:-**

Buffering,
prefetching,
Disk arm scheduling to minimize movement.

Cache → Main memory → Flash → Mag disk → Opt disk → Tape
Primary | Secondary

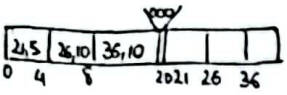
Platter has tracks. Tracks has sectors.

Seek Time : Avg = $\frac{1}{2}$ Worst } Same for Rotational

Wear levelling → done because erase makes storage unreliable

MTTDL = $(MTTF)^2/2MTTR$ = $M/n(n+1)\lambda^2$

RO: Striping R1: RO+ Mirroring R5: Parity
R6: P+Q Redundancy



Too block reads + 2 writes for a read W

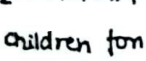
Heap File : Free-space map → Free frn = Value/2^n

Sequential : ordered by search key

Ch-14 Indices Ordered I : stores search key in sorted order

Clustering : S.K. defines sequential order
Non-clus. : diff from seq. order

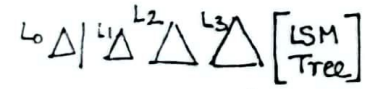
Dense : For every S.K. else Sparse

B+-Tree : 

$\lceil n/2 \rceil$ and n children for non-leaf
$\lceil (n-1)/2 \rceil$ and $(n-1)$ values for leaf

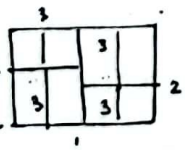
Complexity : $\lceil \log_{\lceil n/2 \rceil}(N) \rceil$

* Bulk Loading → 1. Sort entries first
2. Bottom up B+ Tree constr.

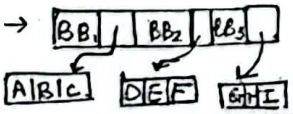


Buffer Tree → Drawback: More random I/O
→ less overheads on query

* K-d trees → partitions one dim into 2.



R-trees →



** Pipelining

open() closed() next()

file scan: Initialise
file scan: output next tuple
Merge join: sort relations
Merge join: continue with merge
demand driven (lazy)
Producer driven (eager)

Name - Akash G
Roll No - 190050038

Linear Search $t_s + b_r * t_T$
Clustering, Eq $(h_i+1)*(t_T+t_S)$
Clustering, N-K $h_i*(t_T+t_S) + t_S + b*t_T$
Secondary, Eq $(h_i+1)*(t_T+t_S)$
Secondary N-K $(h_i+n)*(t_T+t_S)$

n = no. of records fetched

Comparison == Eq. on N.K.

Nested Join → $b_T = n_r * b_s + b_r$ (Transfer)
Seek ← $b_S = n_r + b_r$

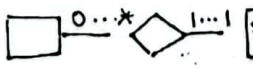
Block-Nested
$T = b_r \times b_s + b_r$
$S = 2b_r$
Best-Case → $b_r + b_s$: T
2 seeks

Merge Join T: $b_r + b_s$ *b_b = buffer blocks for each rel n.
$\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil$ ← seeks

Sorting:
* BT → $b_r\left(2*\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r/M) \rceil + 1\right)$
S → $2\lceil b_r/M \rceil + \lceil b_r/b_b \rceil \left(2*\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r/M) \rceil - 1\right)$

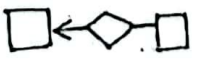
* Pre-Midsem

rank() over (partition by dept_name order by GPA desc).



← Student can have only 1 advisor

← Weak Entity set.



one to many cardinality

Conversion from E.R.→
weak entity → Same attr + PK
Composite → flattened
MVD → separate schema
Many to Many → separate → PK of both
Many to One → add extra attr. to many side (PK)

Lossy decomposition → $r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

* f.d $\alpha \to \beta$
$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

or $R_1 \cap R_2 \to R_1$, $R_1 \cap R_2 \to R_2$ } Lossless decomp.

BCNF : $\alpha \to \beta$ trivial or α is superkey
decomp : $\alpha \to \beta$ cause violation
$\alpha \cup \beta$ & $(R - (\beta - \alpha))$

3NF-algo
→ F_c = canon. cover
if $\alpha \to B$ in F_c and no schema contains $\alpha B \to R_i = \alpha B$
if no schema contains candidate key R_i = C.K.
→ remove subset relations

Sharding (Hadoop)
↓
partitioning data into multiple datasets + scalability

3NF : $\alpha \to \beta$ is trivial OR α is superkey OR each attr. in $\beta - \alpha$ is contained in a candidate key for R.

Armstrong axioms : $\beta \subseteq \alpha \Rightarrow \alpha \to \beta$ Reflexive
$\alpha \to \beta \Rightarrow \alpha\gamma \to \beta\gamma$ Augment.
$\alpha \to \beta, \beta \to \gamma \Rightarrow \alpha \to \gamma$ Transitive

MVD → $\langle y_1, z_1, w_1 \rangle$ $\langle y_1, z_2, w_2 \rangle$
$\Rightarrow \langle y_1, z_1, w_2 \rangle$, $\langle y_1, z_2, w_1 \rangle$

Fourth Normal Form :
$\alpha \to\to \beta$: $\alpha \to\to \beta$ trivial [$\beta \subseteq \alpha$ or $\alpha \cup \beta = R$]
α is superkey for R
4NF ⇒ BCNF ⇒ 3NF

$TF(d,t) = \log(1 + n(d,t)/n(d))$
$IDF(t) = 1/n(t)$
$r(d,Q) = \sum TF(d,t) * IDF(t)$

Page Rank : $P[j] = \frac{\delta}{N} + (1-\delta) * \sum (T[i,j] * P[i])$
$P[i] = 1/N$ set initially

Precision → % of returned results correct
Recall → % of relevant results returned

Map-Reduce :
map(string record) :
for each word in record :
emit (word, 1)
reduce(string key, list value)
word = key int count = 0
for value in value_list
count += value
output (word, count)

Algo for BCNF :
result = (result) U (Ri - B) U (αB)

4NF decomp.
result = (result − Ri)
U (Ri − B) U (αB)

** SQL →
count (case when A = 'c' then 1 else 0 end)

OLAP
Measure attributes
Dimension attributes

* Cookie - sent by server on first interaction
* sent by browser on further

M.V.D Transitivity
$\alpha \to\to \beta$ $\beta \to \gamma$
$\alpha \to\to \gamma - \beta$

not by me

Query Optimisation

$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 \equiv E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$

$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) \equiv (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$

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

$\Pi_{L_1 \cup L_2}(E_1 \bowtie E_2) \equiv (\Pi_{L_1}(E_1)) \bowtie_{\theta} (\Pi_{L_2}(E_2))$

$\equiv \Pi_{L_1 \cup L_2}(\Pi_{L_1 \cup L_3}(E_1) \bowtie_{\theta} \Pi_{L_2 \cup L_4}(E_2))$

L_3 = att of E_1 in θ but not in $L_1 \cup L_2$

$2(n-1)!/(n-1)!$: Different join orders

3^n Time : DP Space : 2^n

$n2^n$ Time left-deep Space : 2^n

Opti. cost budget : Stop opti. early

Plan caching : reuse prev. plan

$\sigma_{A \le v}(r) \rightarrow n_r \times \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$

Selectivity = s_i / n_r

Conj $\rightarrow n_r \times \left(\frac{s_i}{n_r}\right)^n$ Disj $\rightarrow n_r \left[1 - \left(\frac{n_r - s_i}{n_r}\right)^n\right]$ ($i \dots n$)

$R \cap S = \phi$: $r \bowtie s \equiv r \times s$

= Key of R : $\le s$

else : $n_r * n_s / V(A,s)$

$G^{\gamma}_{A}(r) = V(G,r)$ count

$r \bowtie S \rightarrow$ if all attr. are from r

$V(A, r \bowtie S) = \min(V(A,r), n_{r \bowtie S})$

if A contains A_1 from r, A_2 from s

$= \min(V(A_1,r) * V(A_2 - A_1, s), V(A_1 - A_2, r) * V(A_2, s), n_{r \bowtie s})$ (unique values)

$\min/\max = \min(V(A,r), V(G,r))$

** Transactions / additional Opt.

Semijoin $\rightarrow r \ltimes s$, Anti-Sj : $r \bar{\ltimes} s$

select A from $r_1 \dots r_n$ where P_1 & exists(select * from $s_1 \dots s_m$ where P_2^1 & P_2^2)

$\equiv \Pi_A(\sigma_{P_1}(r_1 \dots r_n) \ltimes_{P_2^2} \sigma_{P_2^1}(s_1 \dots s_m))$

involves correlation variables

Material views: (Join) $v^n = v^o \cup (i_r \bowtie s)$

$v^n = v^o - (d_r \bowtie s)$

For projection, Keep count

changes persist even if system fails

Atomicity / Consistency / Isol^n / Durability

Conflict. s

View serializable

1. Same txn must read init. value
2. Same txn must write after reading by same txn's across 2 schedules → eg. first read/
3. Same txn for final write | V.S = C.S. + Blind writes

1) Serializable 2) Repeatable read 3) Read committed 4) Uncommitted

* Concurrency Control

Two-Phase Locking (recoverable) → (serializable)

Grow Phase / Shrinking Phase

1) Strict → hold all x-locks till abort
2) Rigorous → hold all locks "

Graph-Based Protocol

Partial ordering : $D = \{d_1, \dots d_n\}$ → database graph (only x-locks)

→ Serializable + No deadlock + shorter wait time

No recoverability or Cascade freedom

→ cannot relock a data

Deadlock Prevention : wait-die (non-pr) / wound-wait (c-emptive) → older txn waits

Wound-wait : older wounds never younger txn waits

Multi-granular : DB → area → file → record.

IS : intention-shared / IX / SIX

X → false with every other lock

SIX → true only with IS

S → false with IX

↳ info enough to build matrix

Phantom → Updating / Adding Reads tuple that satisfy pred

Index-Locking Protocol → → lock leaf-nodes

Next-Key locking Protocol → → lock values satisfying lookup → lock next-key value

TSO-Protocol

Read : $TS(T_i) \ge W\text{-}t(Q)$

$\Rightarrow Rt(Q) = \max(Rt(Q), TS(T_i))$

else roll-back

Write : $TS(T_i) \ge W\text{-}t(Q)$ & $\ge R\text{-}t(Q)$

$\Rightarrow Wt(Q) = TS(T_i)$

else rollback

$\Rightarrow$ Seriali. + No deadlock

* may not be recoverable → Soln : writes performed at end as atomic actions

Thomas Write Rule → if $TS(T_i) < W\text{-}t(Q)$ then ignore the write operation

⇒ View-serializable.

** Validation-Based

1. Read / Ex phase 2. valid. phase 3. Write

Validation : finish TS(Ti) < start-TS(Tj)

OR start TS(Tj) < fTS(Ti) < vTS(Tj)

Ti & written items don't intersect with Tj (read)

** MVCC-TSO

$\max(RT(Q_K), TS(T_i))$

Q_K : largest t.s. (write) $\le TS(T_i)$

Read : value of Q_K returned $RT(Q_K) =$

Write : $TS(T_i) < R\text{-}t(Q) \rightarrow$ rolled-back

$TS(T_i) = W\text{-}t(Q_K)$: overwritten otherwise → new version of Q_i initialised, to $TS(T_i)$

** MVCC-TPL $Wt(Q_1) = \infty$

Update : Follow rigorous TPL ↑

First write by T_i results in Q_i [new]

$TS(T_i) = ts\text{-}counter + 1$ $W\text{-}t(Q_i) = TS(T_i)$

$ts\text{-}c = ts\text{-}c + 1$

No locks needed ← Read-Only :

* $TS(T_i) = ts\text{-}counter$ serializable

* Follow MVCC-TPL for reads.

Commit processing info

MVCC drawbacks : Extra + space Tuples Overhead for ↑

Recovery System

Logical error (internal) (eg. deadlocks) / System error

Shadow Copying / Paging : update ptr ← edited block

<Ti start> <Ti, X, V1, V2> <Ti commit>

immediate modif. → uncommitted modifi. / deferred modif. → at time of commit

undo(Ti) → <Ti, X, V> ∈ <Ti abort>

<checkpt L> : List of txn active.

Rollback → <Ti, Xj, V1, V2> ⇒ <Ti, Xj, V1>

<Ti start> ⇒ <Ti abort>

Redo phase → i) <checkpt L> : set undolist to L.

2) Scan forward & redo actions

3) <Ti start> → undo list

4) <Ti com/abort> - remove from list

Undo phase → Rollback undo-list

No-force policy + Steal policy

↳ committed may not be written + uncommitted may be written

* Fuzzy Checkpt → Note list of modified buffer blocks (M). Output M to disk (don't allow updts)

Remote Backup System → One-Safe ← Two-safe → Two-very-safe