Lec 19 Multi Version Concurrency Control (MVCC)

Isolation Levels

Can be specified at SQL level to control concurrency

The levels: ↑ isolation level ↓ performance

- serialisable — no phantoms, no dirty reads, all reads repeatable
  - get all locks first, index lock, and strong strict 2PL
- repeatable read — allow phantoms
  - same as above but without index lock
- read committed — allow phantoms and unrepeatable reads
  - same as above but release S lock immediately
- read uncommitted — allow all three
  - same as above but no S lock

Other things
- strict serialisable (e.g. Google Spanner) ← serial by commit time order
- snapshot isolation
- cursor stability

MVCC

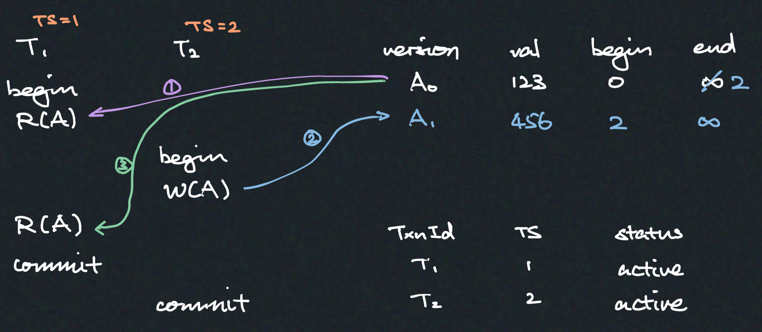
DBMS keeps physical version of logical object in DB

Write: make new version
Read: find the correct version

Benefits: W doesn't block R
R doesn't block W] they can find the right version and time travel

Snapshot: like repo at commit

Ex.



version	val	begin	end
A_0	123	0	~~∞~~ 2
A_1	456	2	∞

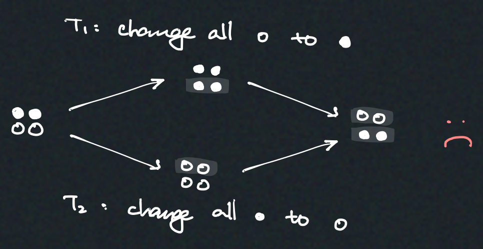
TxnId	TS	status
T_1	1	active
T_2	2	active

Each record can have multiple version chains, but only one writer at the end

Snapshot Isolation

Each tuple has own timeline
Snapshot cuts through timeline

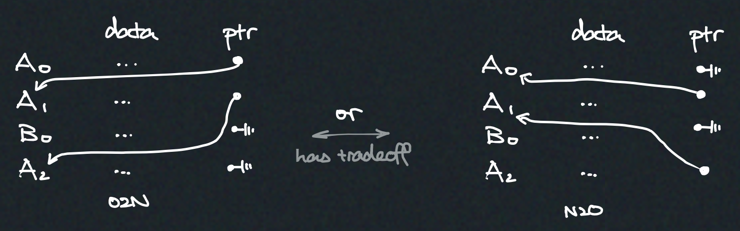
Write skew anomaly



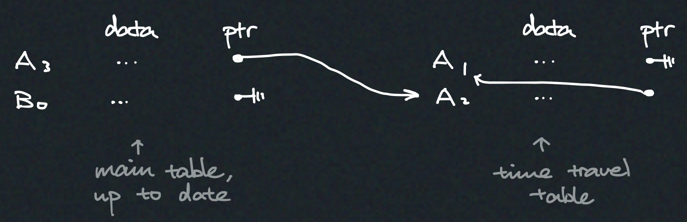
Version Storage

Add field to tuple, pointing to version chain ?

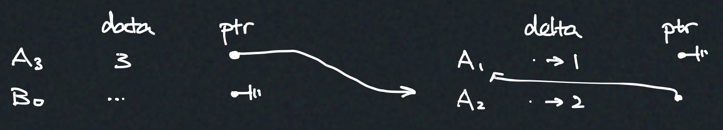
- Append only storage



- Time travel table



- Delta storage



Garbage Collection

Remove reclaimable versions
└ those older than earliest active exns

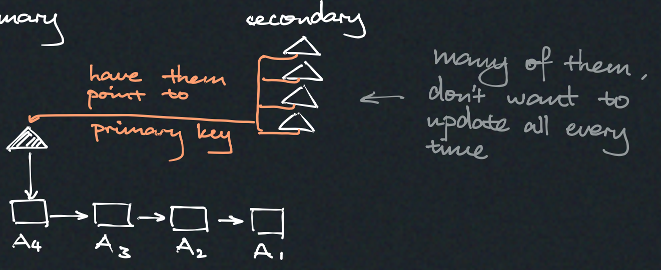
- Tuple Level
  - Background Vacuuming — remove those tuples with some periodic background job
    → Keep dirty page bit for performance
  - Cooperative Cleaning — each txn identify expired versions
- Txn level — transactions keep list of old records, and sends it to vacuum upon commit

Index management

Primary key: point to version chain head

Secondary ... hmm

- Logical ptr



← many of them, don't want to update all every time

- Physical ptr

Duplicate key ... may cause problem
Delete ... can have dead chains