# Final prep

## *Notes:

- Query → data retrieval.
- Transaction → some data to be read/written in the database.
- Info stored by DBMS; Metadata
- Tuple → sequence / list of values.
- atomic → indivisible. / can't be broken down.
- Key → Uniquely identify tuple
- SuperKey → set of Keys.
- Minimal SuperKey → Candidate Key.
- Candidate Key: Several distinct attributes.
- Primary Key → Candidate Key → identifies tuples → property of entire relation.
- Foreign Key → link PK. to its relation

## * Attributes:

Simple → atomic

Composite Attributes → divided into subparts.

Multivalued: phone#, DOB

Derived: inferred / calculated from another attribute → age from DOB

## ✱ Normalisation:

- DB free of anomalies

1NF: only atomic values / single values / set of values.

Resolution

- eliminate repeating groups.
- seperate table for each set of related data.

2NF:

- every non prime attribute A in R is fully functionally dependent on PK of R.

Test: LHS attributes = PK.

- If PK = single attribute → no need for test (provided there are no partial dependencies on CK).

PK1 PK2 NPA1 ✓ fully depends on PK.

└──────┘
1

PK1 PK2 NPA2 X partially dependent on PK.

Resolution:

- NPA are associated only with the part of the PK they are dependent upon. (fully functionality)

| PK1 | PK2 | NPA1 |
|---|---|---|

| PK1 | PK2 | NPA2 |
|---|---|---|

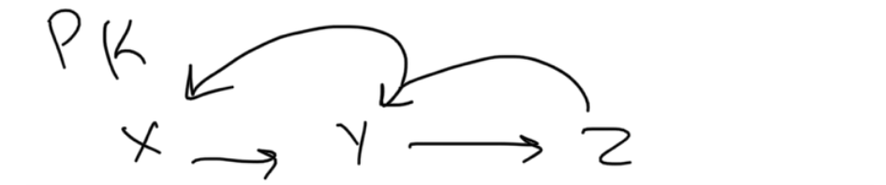
3NF: Transitive Dependency

$x \rightarrow Y \rightarrow Z$

hence

$x \rightarrow Z$

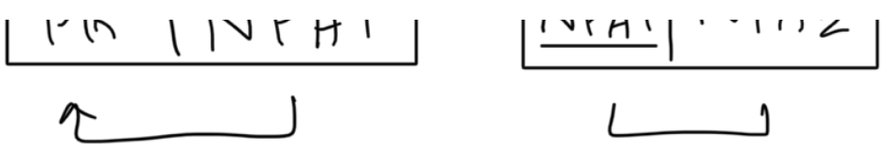
- No NPA is trans. dependent on PK




| PK | NPA1 | NPA2 |
|---|---|---|

| PK | NPA1 |
|---|---|

| NPA1 | NPA2 |
|---|---|



2NF → PK contains multiple attributes → no partial dep.

## ✱ Transaction Processing:

- The lost update problem.
  ↳ an updated value gets lost as it is not written.

| $T_1$ | $T_2$ |
|---|---|
| 80 r(X) | |
| 75 X := X-N | → lost update. |
| | r(X) 80 |
| | X := X+N 85 |

- The temporary update problem:

| | $T_1$ | $T_2$ |
|---|---|---|
| | r(x) | |
| T | X = X-N | |
| I | | |
| M | W(x) | |
| E | | r(x) |
| | | X = X+M |
| ↓ | abort. | W(x) |

- Occurs when 1 transaction updates an item & fails → but updated item is used by another transaction b4 item is changed or reverted back. to its last value.

## - Incorrect Summary:

- Transaction calculates some values b4 they are updated and others after they are updated.

example:

| $T_1$ | $T_3$ |
|---|---|
| | Sum = 0 |
| | r(A) |
| | S = S + A ; |
| r(x) | |
| x = X - N | |
| w(x) | |
| | r(x) → reads x after n is subtracted & y b4 n is added hence wrong summary |
| | S = S + X |
| | r(Y) |

$S = S + Y$

$r(Y)$
$Y = Y + N$
$w(Y)$

Unrepeatable read:
- $T_1$ reads the same item twice but item is changed by $T_2$ b/w 2 diff reads.
  example → Air line seats.

★ Transactions: → Should either be performed in its entirety or not at all. (Atomic Unit)

A tomicity
Consistency
Isolation
Durability.

→ $T_1$ should take DB from 1 consistent state to another if it is completely executed from beg. to end. w/o interfering w other transactions

Each $T_n$ is unaware of other trans. executing simultaneously.
Transact must operate without interference by any other T executing Concurrently.

Changes to a Systems persist even if there are system failures.
Once a T completes successfully all its updates in the DB persist even if there is a system failure.

## * Storage Structures

volatile → does not survive system crashes → cache memory.

Non volatile → survives system crashes. → secondary storage devices

Stable → data is never lost.

Aborted → T does not execute successfully
↳ if changes have been undone → T has been rolled back

Committed → T completes execution succesfully
↳ effects can't be undone.

## ✱ Transaction states:

Active → stays in this while executing.
Partially → after final has been executed.

- Serial Schedule: (→ order of execution.) Instructions belonging to one transaction appear together in that schedule. → always consistent
- Concurrent aren't always consistent.

if concurrent = Serial → serializable.

(→ only on same var.)
r-r → no conflict
r-w, w-r, w-w → yes conflict.

Swapping: If I and J (instructions of diff transactions) do not conflict, we can swap them to produce a new schedule S.

- If swapping can convert S into

serial → conflict equivalent

View Serializability: S may not be Conflict Serializable but it may produce the same view as a serial schedule.

## * Concurrency Control:

- implemented through locks.

Shared → read.

Exclusive → read & write.

| | S | X |
|---|---|---|
| S | T | F |
| X | F | F |

Delayed Unlocking
↳ Consistent Schedule.

Deadlock. → locks are being requested w/o them being unlocked.
hence → deadlock.

Deadlocks > Inconsistency

-Unlocking too soon may give inconsistent

states. (first in first out basis)

- not unlocking before requesting = deadlock.
  ↳ desirable as they can be handled by rollbacks.

Starvation:

- exclusive locks may not be granted if multiple transactions request shared locks (compatibility matrix)
  ↳ Resolution: Sequential locking.

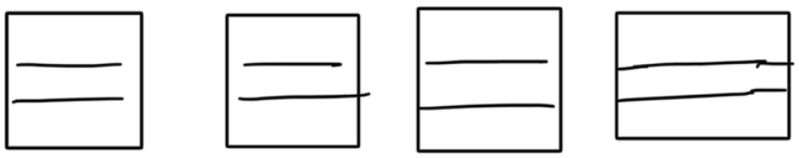
* 2 phase locking protocol:
- ensures Serializability:
  ↳ growing phase → obtains locks → no release.
  Shrink → releases locks → wont obtain, cant issue anymore lock requests.
- Deadlocks can still occur.

* Database backup:

1) Full backup:

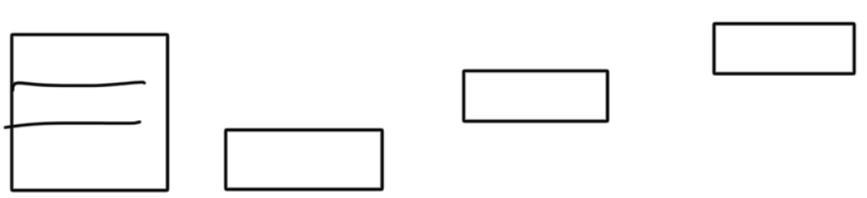
involves copying entire data set. onto external disk.

- alot of space, time, impractical daily.
- better weekly, biweekly.




2) Incremental:

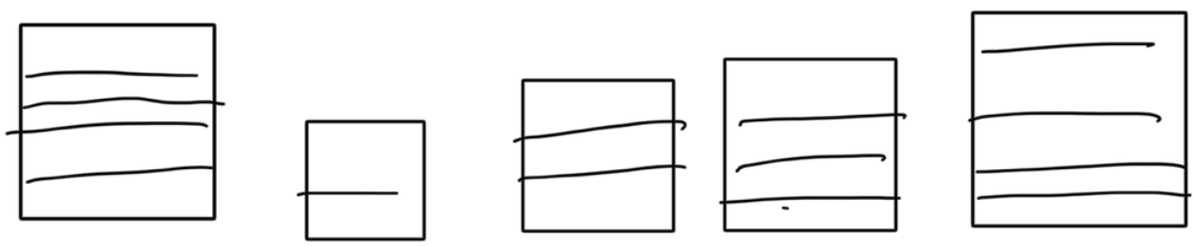
- Stores all files changed since last backup.
- last backup may be full, differential or incremental.
- rescue friendly alterative to full.
- takes up less storage., faster




(full)

v

3) Differential backup:
- updates all data that has changed since the last full backup
- relies on most recent last full backup.
- faster than full
- takes up less space than full.




Summary:

full
Copies entire data

Inc:
full backup + Changes since prev. backup

Diff:
full backup + Changes since last full backup

* Indexing:

- Many queries reference a small portion of records in a file here it is inefficient for system to read every tuple.
- works just like an index in a text book.

* Types:

Ordered: based or souted ordering.

Hash: values are assigned according to hash fution.

* Parameters:

- Access types → item search + range.
- Access time → time taken to find particular data item
- Insertion time: insert a new data item (time taken to search + update).
- Deletion: time taken to delete + update

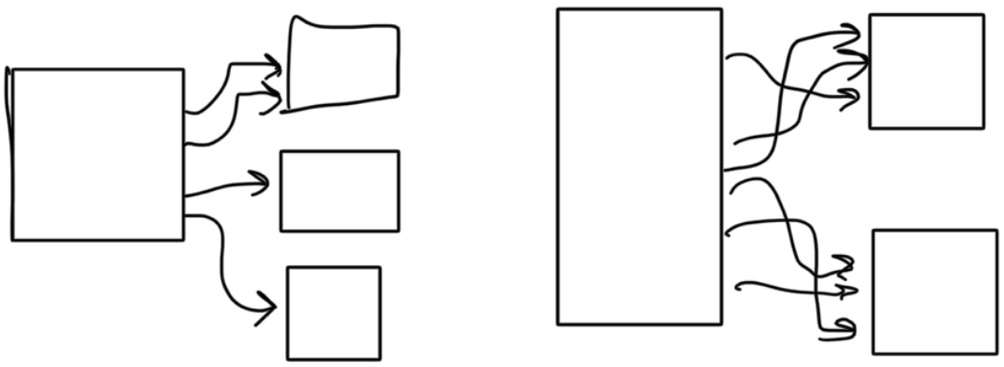
# Ordered Index

Cluster: sequentially ordered → primary Index.

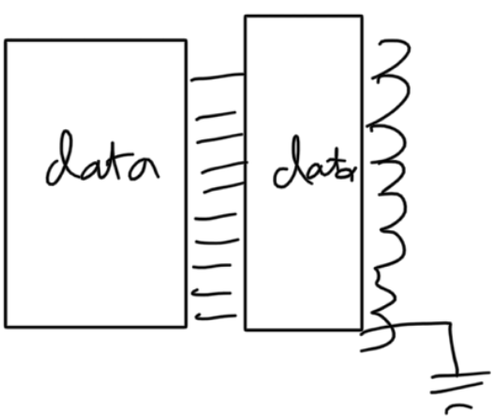
Secondary: order different from sequential → non cluster.

Index Sequential: Clustering Index on the search key.

Clustered:

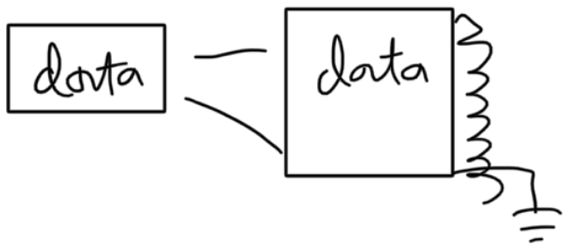


Dense index: index entry appears for every search key value in the file.



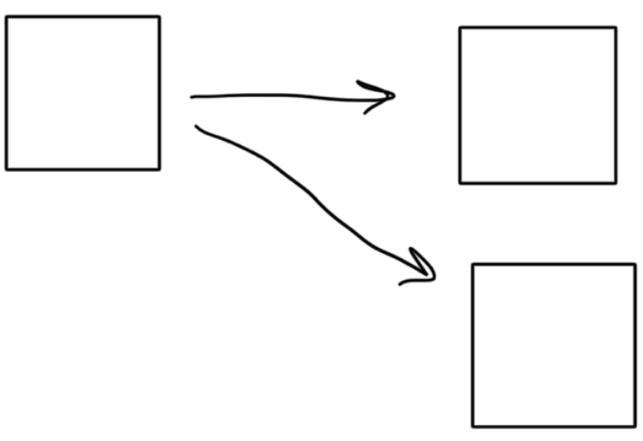

Sparse: index entry appears for only some of the search-key values.

↳ order of relation is in sorted order
i.e clustering index.

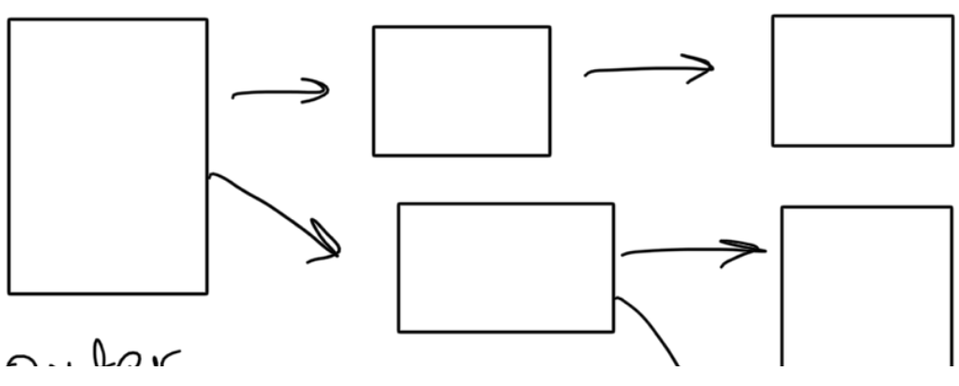



Dense is faster than sparse. (to search)
Sparse require less space.
↳ Sparse index with 1 index entry per block.

Resolution:

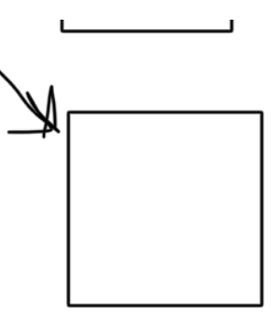


Multilevel indices.



Outer

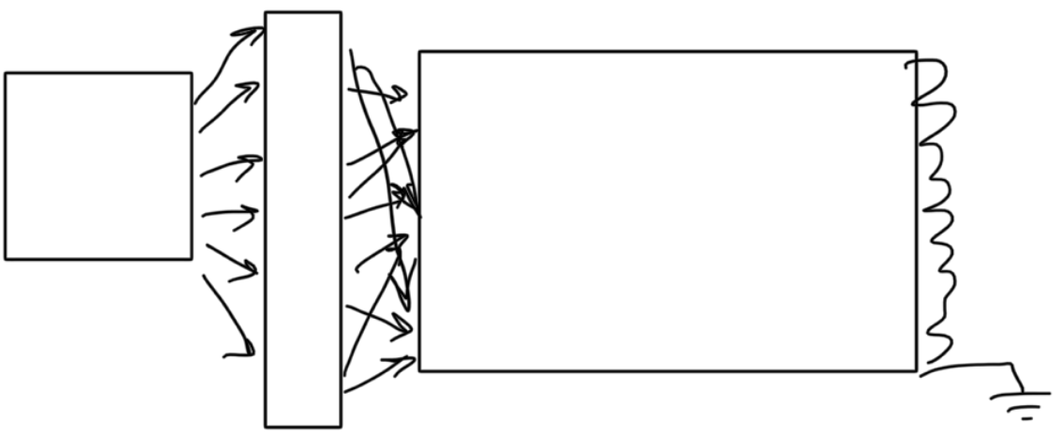
Inner.



2-level sparse.

Secondary:

much like dense: not stored sequentially.



Indexing → efficiently retrieve records from db.

types: dense, sparse, clustered, non clustered.

If index is too big to fit in main memory, several level of indexing may be used.