DBMS - IV

Transactions & Indexes

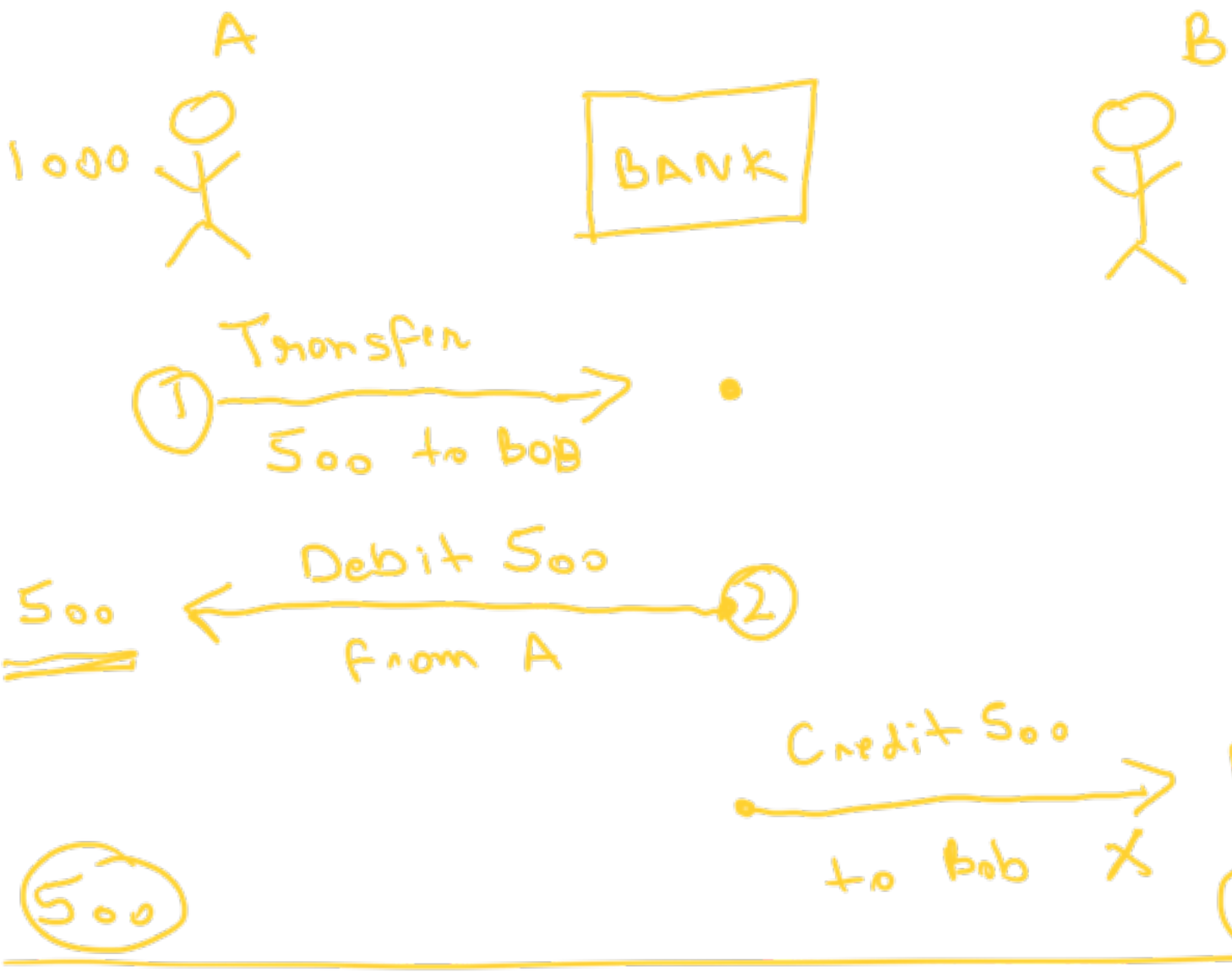
1. Transactions
   - → ACID
   - → InnoDB - ACID
2. Indexes
   - → Type of indexes
   - → Implementation of indexes

Transactions

Banking

- Alice — 1000
- Bob — 1000

Alice $\xrightarrow{500}$ Bob



(1) Low balance

② One operation fails

③ Server crash

---

all or none — atomic

group of related operations that should be executed as one unit of work

→ Debit
→ Credit
} Transaction

BMS

2CA

→ Book a seat
→ Make Payment
} all or none

Transactions - (A)CID

(1) Atomic

→ all or none

DONE vs NOT-DONE

- Simple - undivisible

→ BEGIN Transaction

→ Transfer from A to B

→ Debit from Alice

→ Credit to Bob

begin

Balance := read (ALICE)

if balance < 500 :

ABORT

A = 10

T1

A = 20

rollback

A => 10

ROLLBACK

- Begin
- rollback

begin

balance = ALICE

DEBIT

CREDIT

COMMIT

A = 10

BEGIN

A = 20

COMMIT

A = 10

→ begin

→ rollback

Abort

→ commit)

ABORT — cancel the transaction

Rollback — revert changes
— keep the transaction alive

Bank

① HDFC
→ HDFC down
→ roll back

② ICICI → no run

Abort + roll back → flag

END

HDFC
ICICI

A =10

BEGIN

A=20

error

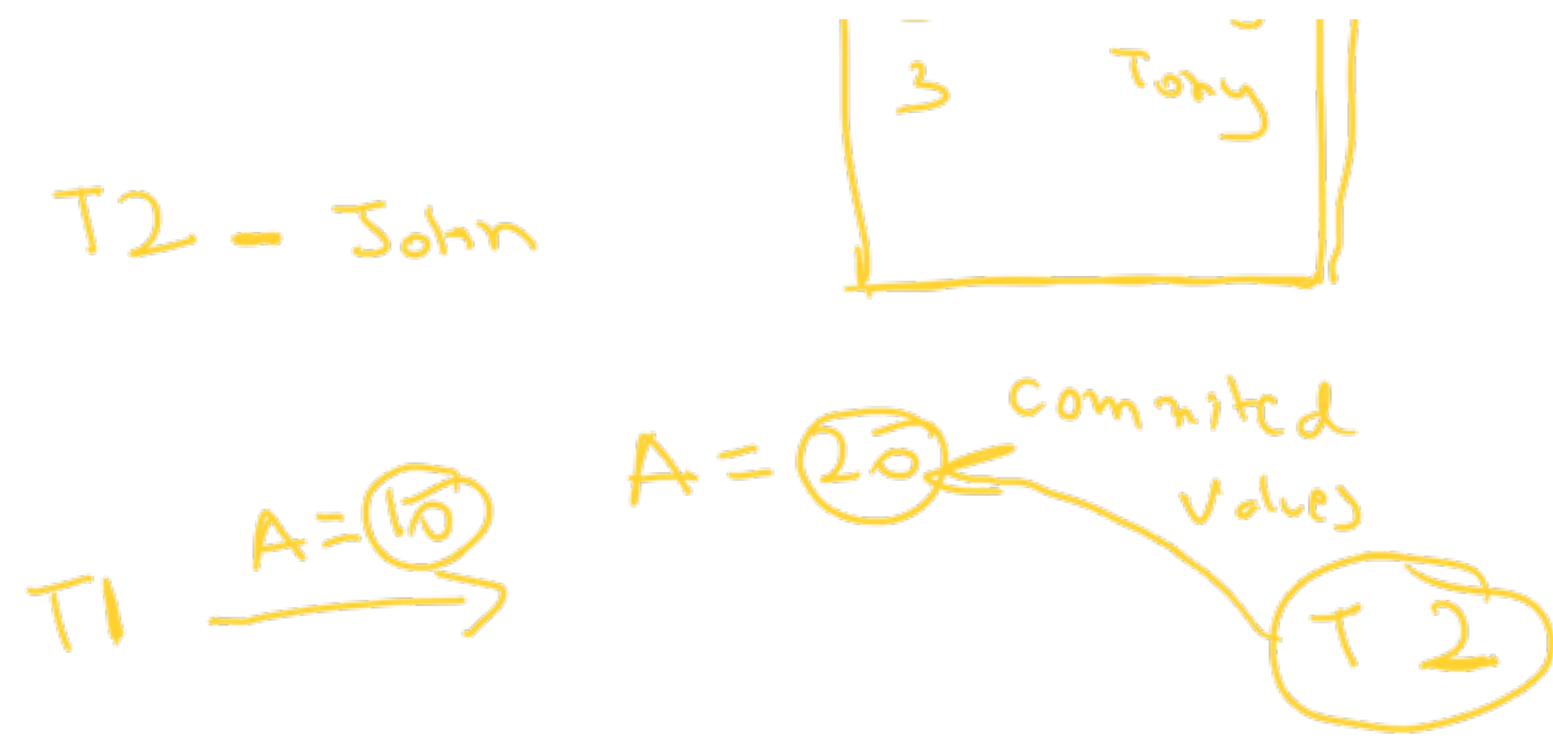
rollback

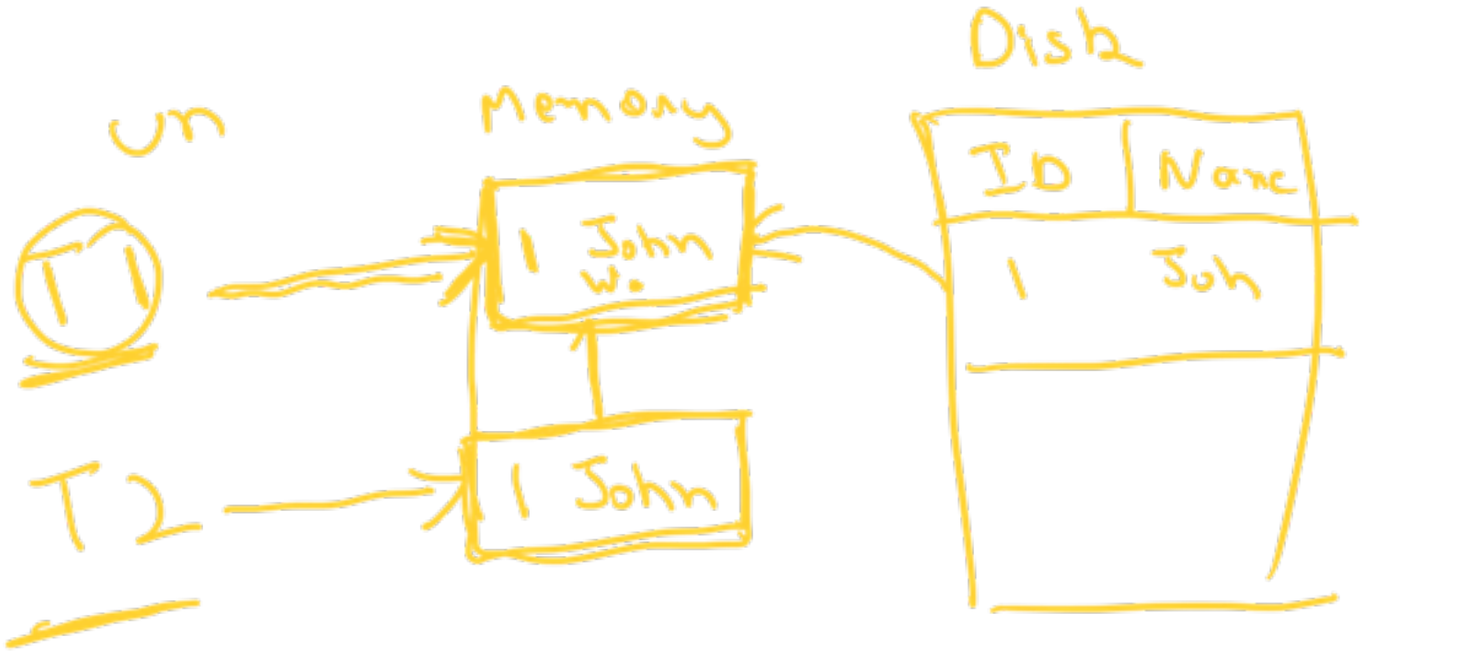
A=10

T1 - John Watson

students

ID	NAME
1	John
2	Mary

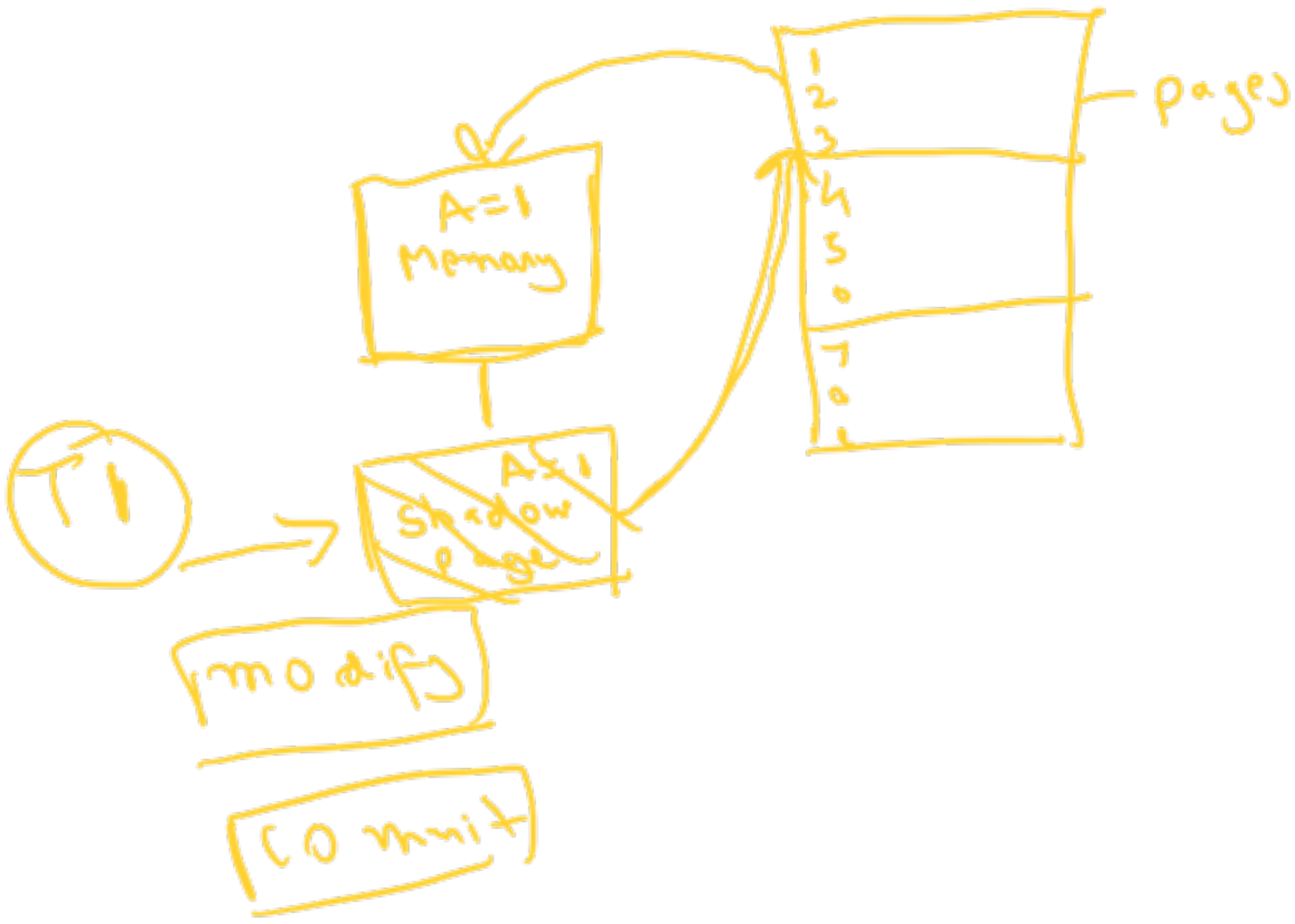

3
Tony
T2 - John
A = 20
Committed
Values
A = 10
T1
T 2


Disk
Memory
ID
Name
1
Joh
1 John
W.
1 John
T1
T2

Commit

Shadow paging — atomicity

↳ 80's — CouchDB



→ memory — copy of a copy

T1 T2 → original

T1 → Shadow page

Transaction → Shadow

Others → memory

InnoDB – Write-ahead logging

Commit

- how we do on undo

redo log

update

A = 20 — ① write in log

② Update in memory

if roll back ① read from log how to revert

WAL

ACID

↳ Atomicity

↳ all or none

↳ BEGIN

↳ ROLLBACK

↳ COMMIT ✓

→ Shadow paging

→ DB keeps a copy of the records in a transaction

→ Transaction uses the copy (shadow) while others use the main

→ WAL (redo log)

→ stores the operations, how to undo/redo

→ In case of failure, DB just uses the log to re-run the operations

UPDATE 10 | UPDATE 20

Consistency

State 1

T1 —

State 2 — Connect

— constraints

— PK, FK

A = 20

Begin

A = "string" Domain

1000 Alice → 1000 bob • 2000

begin

DEBIT → Alice (500)

Failure → 1500

before A+B ⇒ 2000 ⎫

after A+B ⇒ 1500 ⎭

2000 ① constraints are met

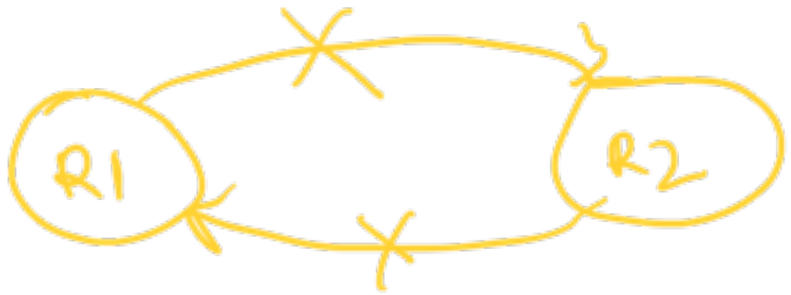
② some bussiness validation

→ A+B = A'+B'

① referential integrity

(2) Application

Isolation



interfere with each other

500
A → B.

T1

bank

T2

sum of all the accounts

set A=500

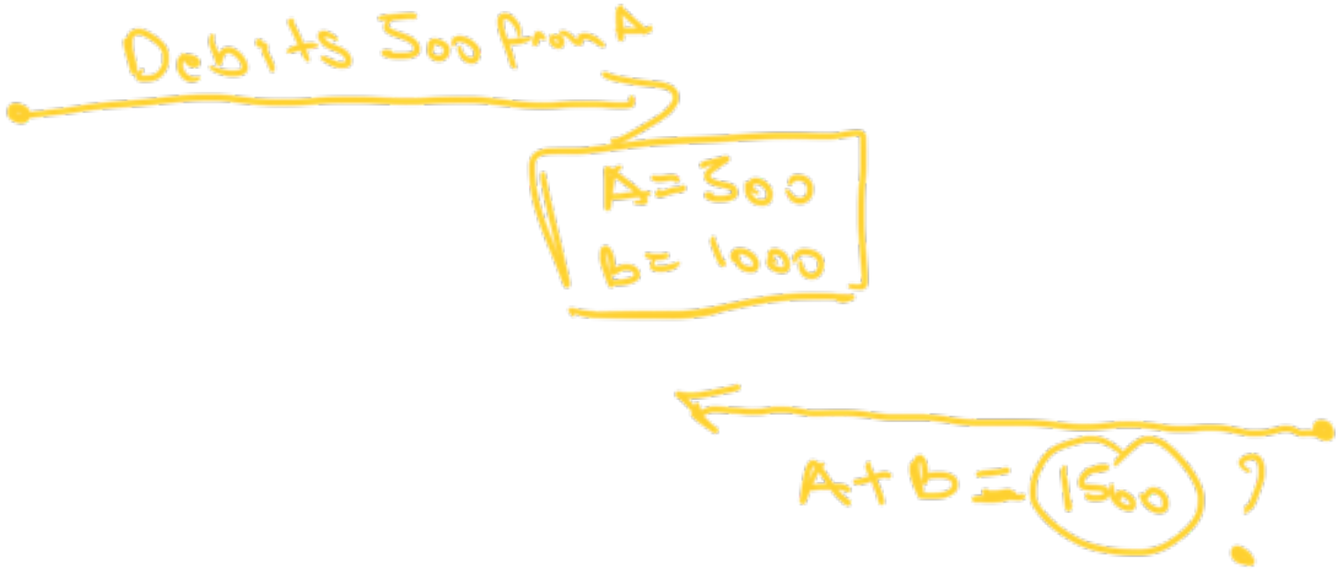
set B=1500

A+B=2000

1000

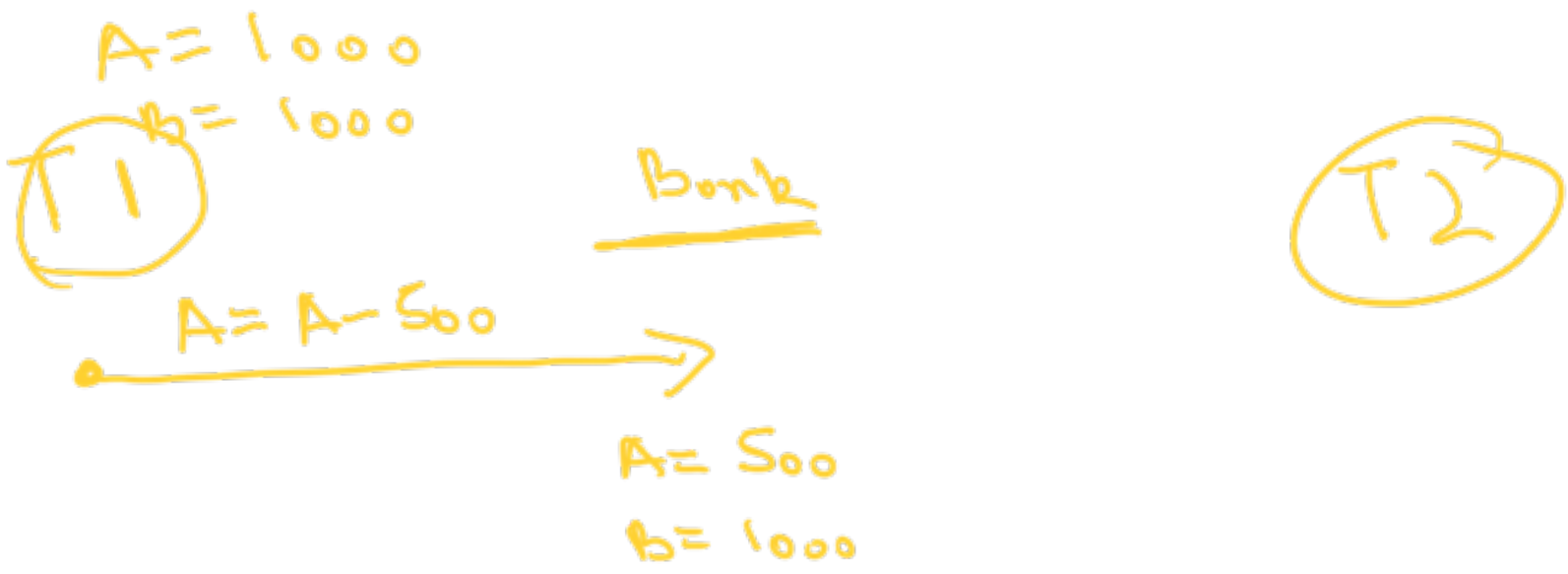
1000

T1 bank T2



Isolation

- helps us to manage how multiple transactions execute



sum

A ← b = 1600

1000 + 1000

= 2000

Isolation levels

T1 → T2 ✓

T1 ⟷ T2

Issues – Phenomena

1 (P1) – Dirty read

↓

updated

0

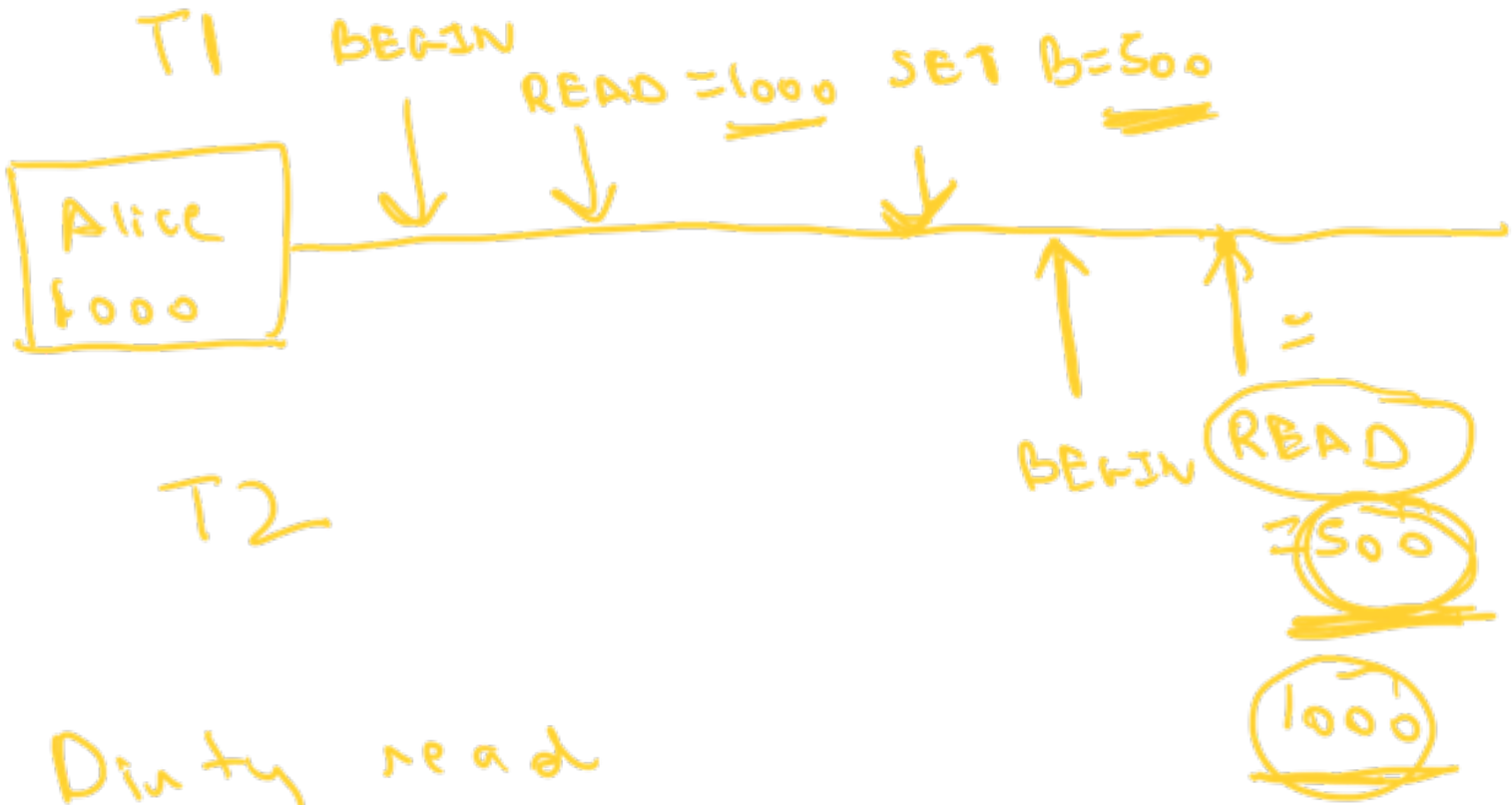
1

↓

P0 - Dirty write

dirty bit

Dirty read (P1)



Dirty read

→ T2 reads uncommitted values

→ T1 - Reads balance as 1000

→ T1 → Updates balance = 500

→ T2 → Reads balance as 500

Dirty read → P1

P0 - Dirty write

T1 → Reads bal. = 1000

T1 → updates = 500

T2 → balance = 500

T1 → Commits (300)

T2 → roll back

Dirty write

P0 (w) P1 (k)

P2 – Non-repeatable reads

P3 – Phantom reads

(P2)

T1 – Read Bal – 1000

T2 – Update Bal – 500

T1 – Read – 500

T1 – 1000
– 500
} non repeatable read

P3 - Phantom (Ghost)

- insertion or deletion

T1 - select* from s = 100 rows

T2 - inserts 1 row

commit

T1 - select* from s = 101

Phantom read

lol

qq

P0 - Dirty write } uncommitted

P1 - Dirty read }

P2 - non-repetable - A = 1000
A' = 500

P3 — phantom — (entry ≥ ten)

Isolation level

1. Read uncomitted — P0, P1, P2, P3
2. Read comitted — No P0, P1
3. Repeatable reads — No P0, P1, P2
4. Serializable

→ Serially

T1

T2

T3

→ No issues (P0, P1, P2, P3)

Concurrency vs performance

Isolation levels

P0
P1
P2
P3

ACID

6:13 - 6:20

10:43 - 10:50

DB - isolation level
   - transaction

My SQL — Repeatable reads

SQL Server — Read Committed

WAL → transactions

— persistent

---

① Cache

② Isolation

Durability

→ your results must be permanent

→ Begin

UPDATE

COMMIT

:

txt lo

:

DB Crashes

permanent — Disk

→ log file

binlog

WAL

binary log

disk

log

Update

↳ Store UPDATE A=10

↳ begin transaction

COMMIT ✗

→ bin log (status)

→ uncomited changes

→ write to disk

→ rollback that operation

Distributed

→ distributed log file

binlog —

transaction —

WAL — redolog

log — periodically cleaned

ACID

Atomic — all or none

Consistency — constraints

— db state correct

Isolation — concurrent transactions

↳ Durability
— permanent after commit

Db — ACID compliant

RDBMS

NoSql — bASE

eventually consistent

In depth

book — Concern

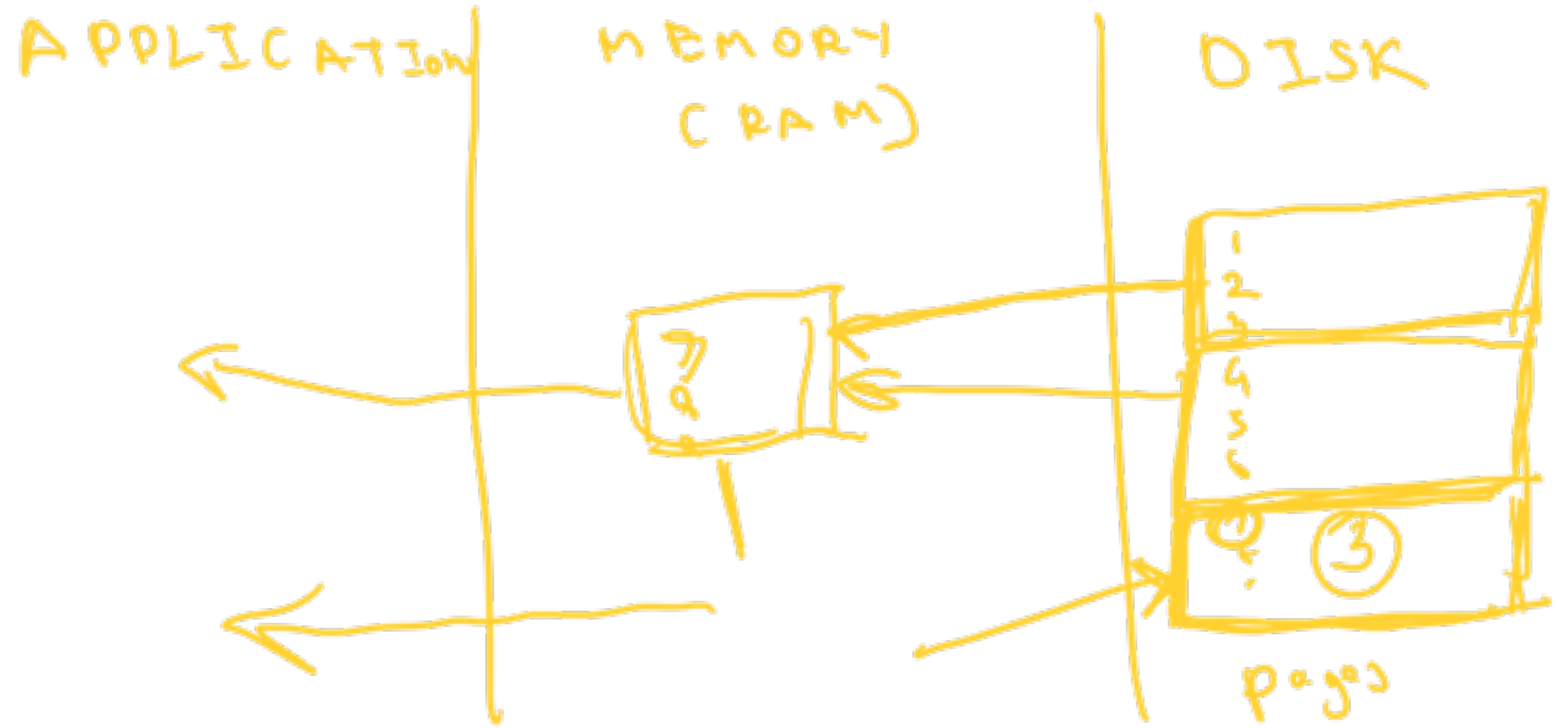
→ backtracking

↳ Panel.tp.2p

→ Index

backtracking- Page 1, 10, 20

Dat abase

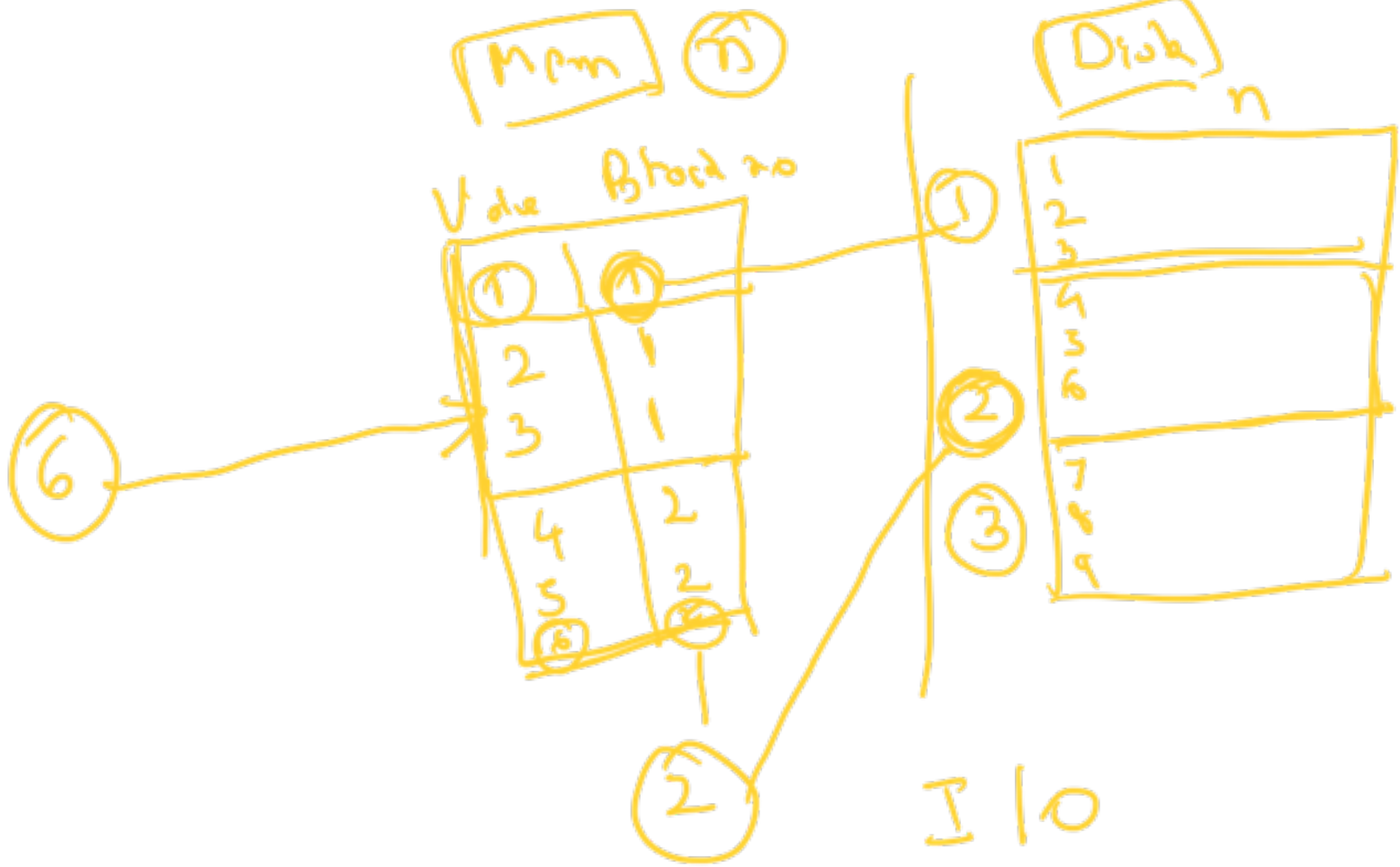


Sequential

Disk → RAM

I/O operation

extremely slow



Index - Data structure
that helps us
reduce I/O operations

Dense index

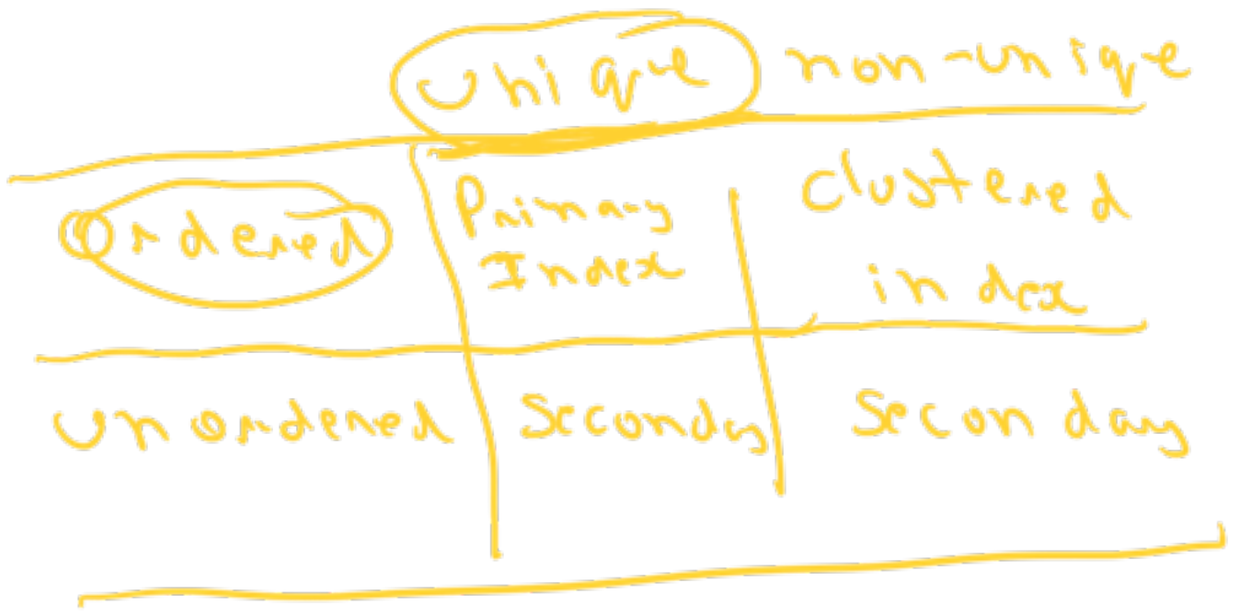
1:1 b/w value : address

⇒ large storage

Type of values

→ unique ✓

→ ordered ✓



PK - Primary Index

Index

→ reduce I/O

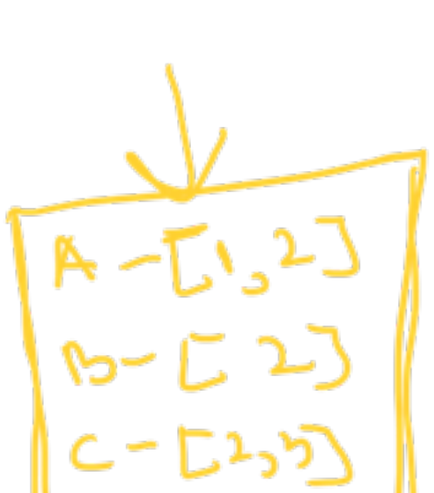
→ increase database

bookkeeping

→ write slower

→ Clustered
→ Secondary

Clustered — Ordered
— non-unique



flag

- trace

1 is flag true?

if yes, fetch next block

① → ② → ③ → ④

Sequential — 4 I/O operations

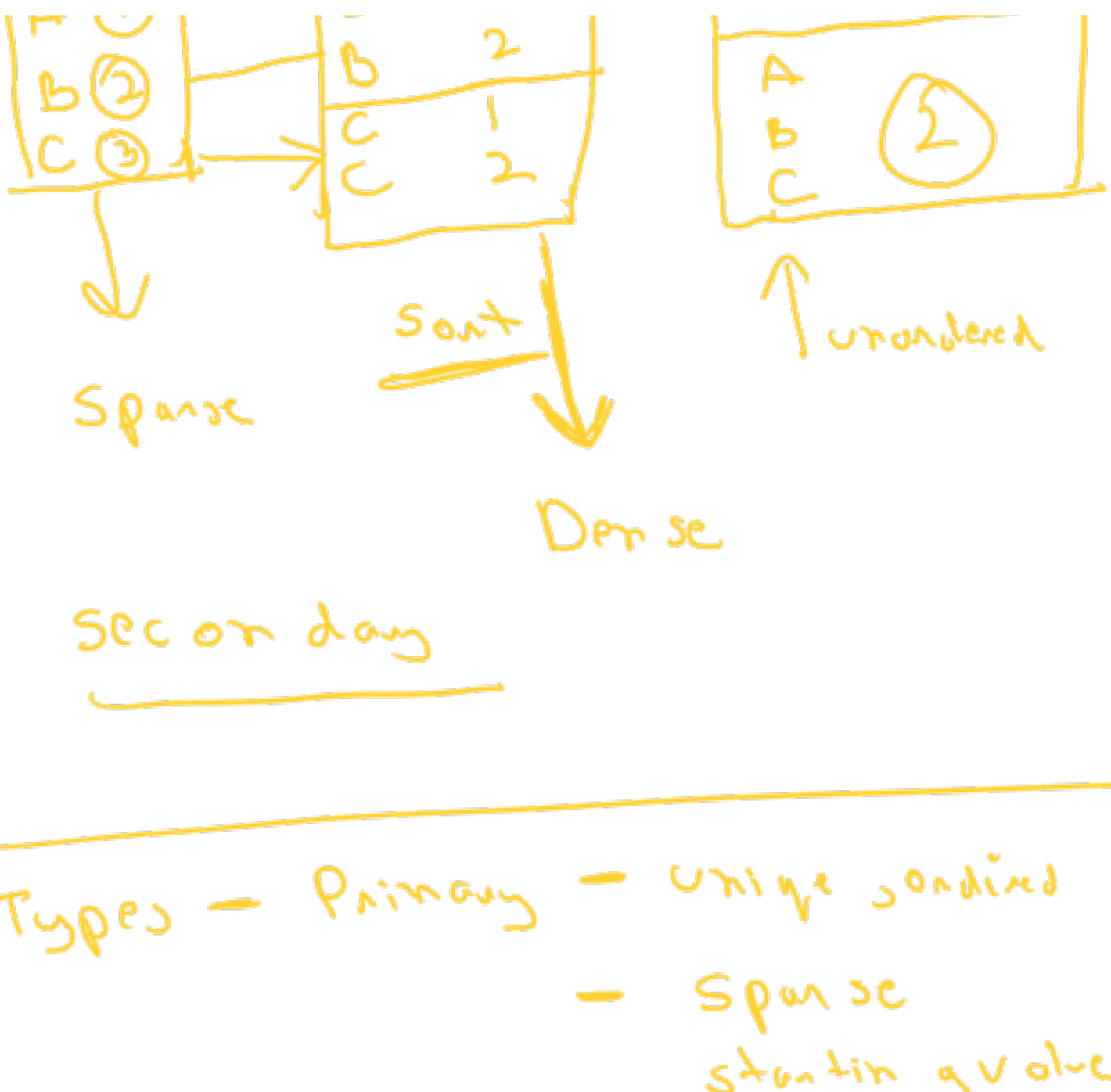
A - 1

b - 2

① Indexes
reduce IO operations

Second day

- Unoided
- non-unique



Secondary

Types - Primary - unique ordered
- Sparse
stores value
address

- clustered - not unique
- sparse + trace

- secondary - unordered
  - two levels
    - 1st dense - sort
    - 2nd sparse
      - lookup

When to use an index

- PK
- select * from - where name / phone

When not to use an index

- null values
- small table }

Transactions - ACID

Indexes - I/O operations

- Sparse / Dense
- Primary
- Secondary
- Clustered

Query execution

- MySQL
- Workbench
- Table plus

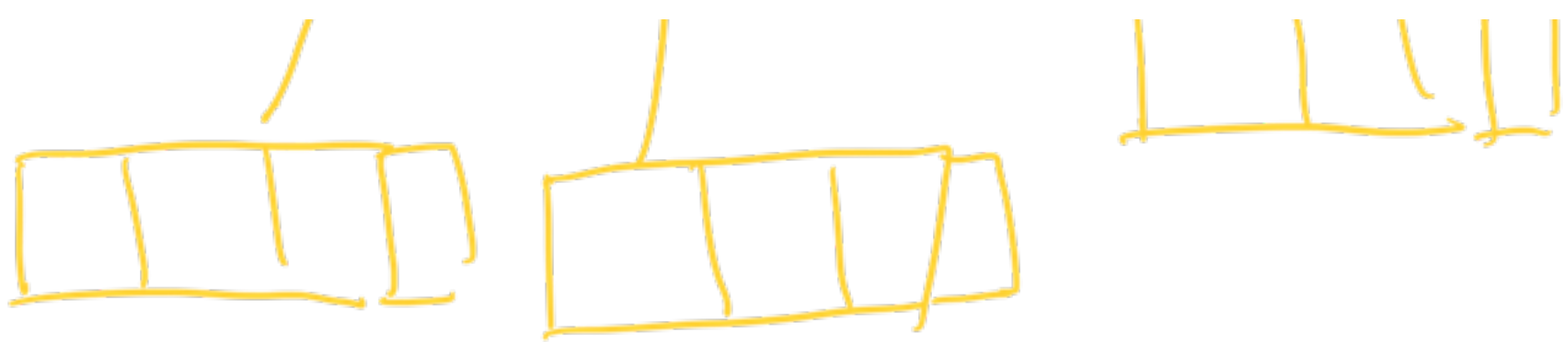
B+ trees

Binary | Balancing trees

updates

B+ trees

- ordered keys
- multiple children



balancing

Implementation details

ACID

|

Consistency

Alice → Bob

A = 1000 bob = 1000

Debit 500 from Alice A + B = 2000

↓

A = 500. B = 1000 — A + B = 1500

↓

Failure

1500 2000

Correct state

1000 + 1000 = 500

business logic A + B + 1500 = A' + B'

Consistency should be met

int — (A) — int

- Concurrency control

⇒ locks

- Read
- Read write
- range

Too many indexes

↳ large memory

↳ writes

Using indexes

FULLTEXT

data gets corrupted

hash()

1 ≠ 2 → corrupt

checksum