DESIGN IRCTC

① MVP

i) search for trains
date, src, dest ⇒ train_id | src_time | dest_time.
↳ categories - (SL, AC-3, AC-2, ..)

ii) User auth / registration

iii) date, src, dest, train_id, class
↳ seats available

iv) date, src, dest, train_id, class
↳ book a seat

v) Payment Gateway → discount

⇒ vi) Notification

vii) PNR ⇒ confirmed?

viii) Cancellation of tickets.

① MVP

② Est. of scale
↳ read / write heav
↳ sharding,

③ Design goals

④ API design
+
Components design

Final Implem

Non-functional ✓
1) Low latency.
2) highly consistent.
3) high throughput

② Est. of scale

$\boxed{10{,}000}$ trains / day

$\underbrace{72}_{\text{\# of seats in bogie}} * \underbrace{15}_{\text{\# of bogie}} * 10{,}000 \simeq$ ~~1~~ 10Mi

10 million seats / day. * 1.1 ✓

= $\boxed{\text{11 M seats / day}}$

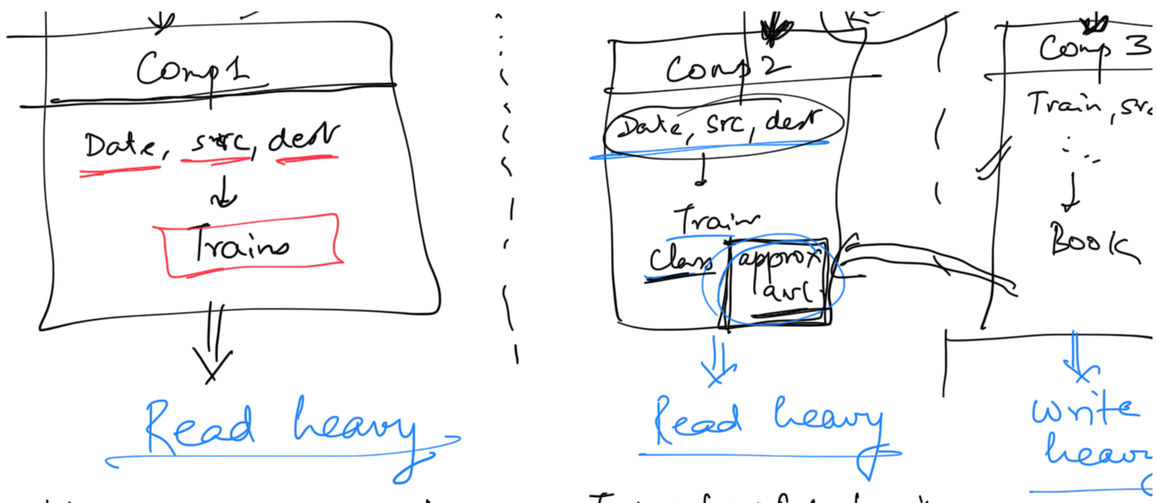
→ booking a ticket
↳ search for train & availability
↳ looking

Highly avl | Read heavy

→ check avail
OR
"check train

← High A Read heavy

Strong Con



No sharding reqd.
$\boxed{\text{In memory}}$

Train_id	date	class	avail
20k	90d	4c	int

$= 720K*(4+8+4+4)$

$= \boxed{14\ MB}$

$\boxed{\text{In Memory}}$

11 million seats

≈ 200 bytes

11 M * 200 bytes ≈ $\boxed{2GB/day}$

$* \boxed{90\ days}$

180 GB

$\boxed{\text{No sharding required}}$

Design goal:

CAP

C3 → HC

C1 + C2 → HA | low

C3:

API:

✓ – $\boxed{\text{bookTicket}}$(date, src, dest, train_id, class, userId, no.of seats)

✓ – cancelTicket(PNR, userId)

passenger_name	userid	train_id	PNR	class	seat_no	id_info	status
128B	8B	8B	8B	4B	4B	8B	4B

JOIN

ACID → Isolation

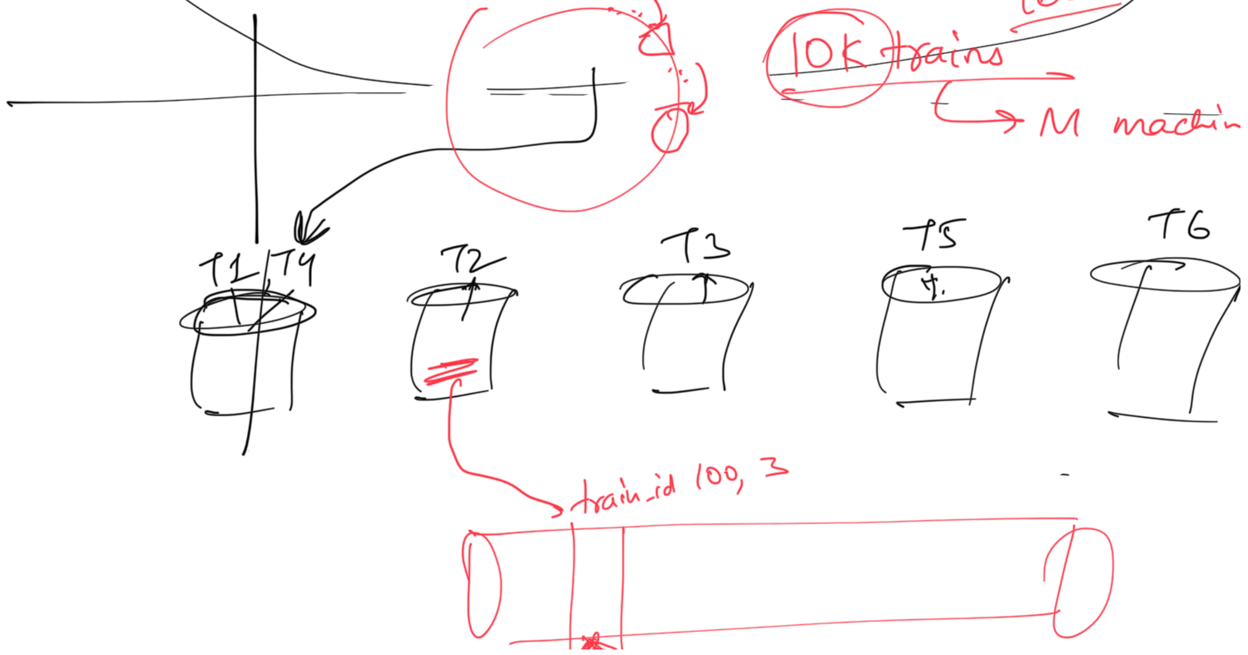
Atomicity

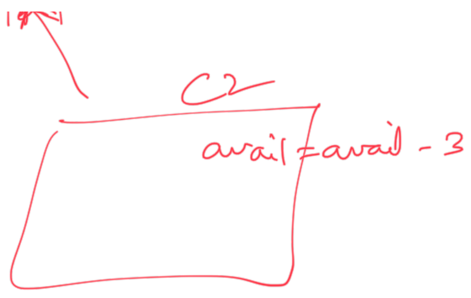
↓ payment

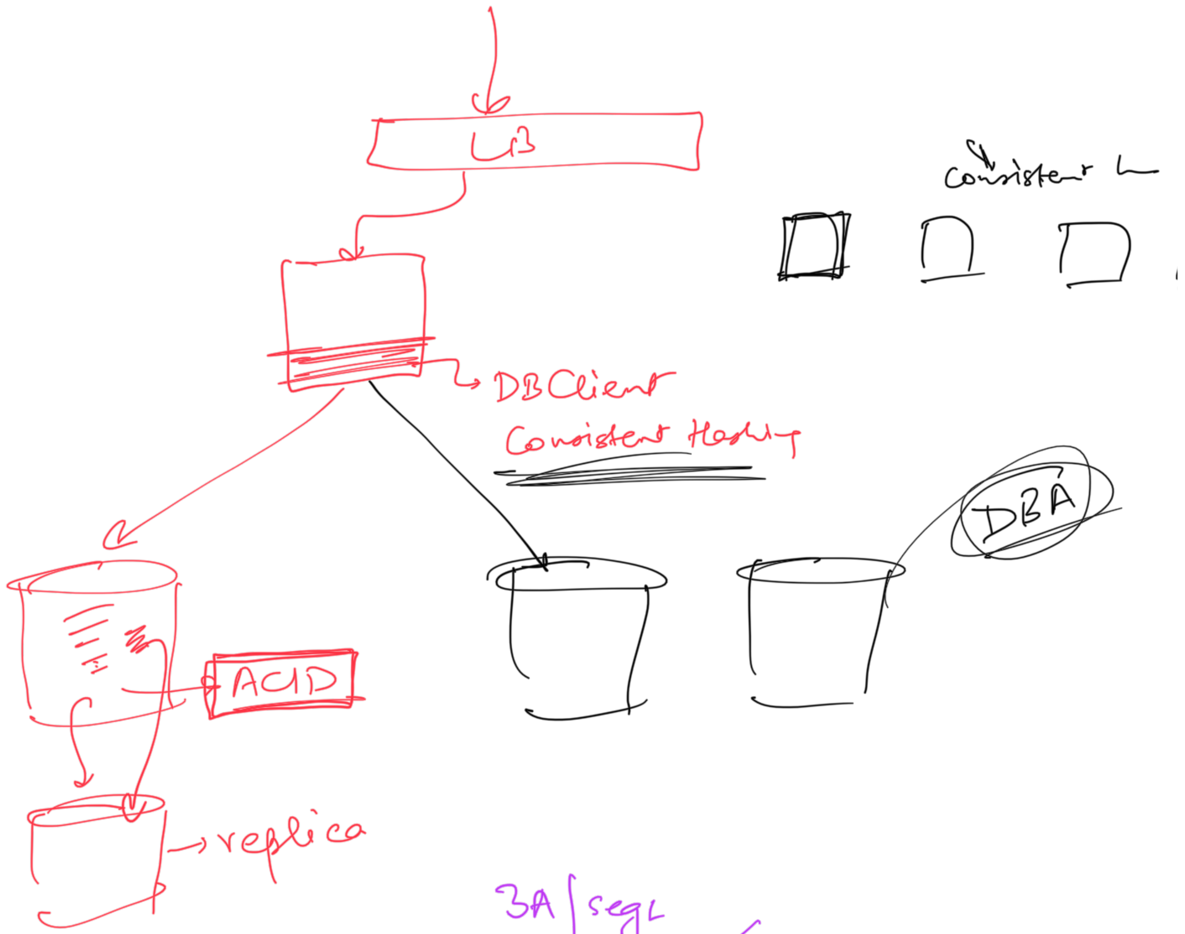
↓ allocate seat

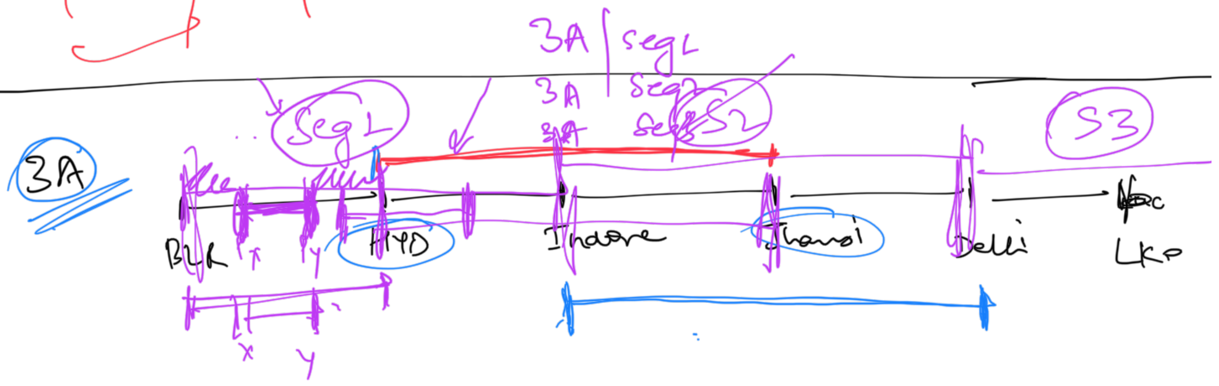
seat → defr
conf.

```
Update SEATS SET USER ~~STAT~~ = userid
WHERE
        train_id = xyz
    ~~AN~~ date = ——
    AND class = 1
~~LIMIT 1~~ AND UserId is NULL
LIMIT 1
```



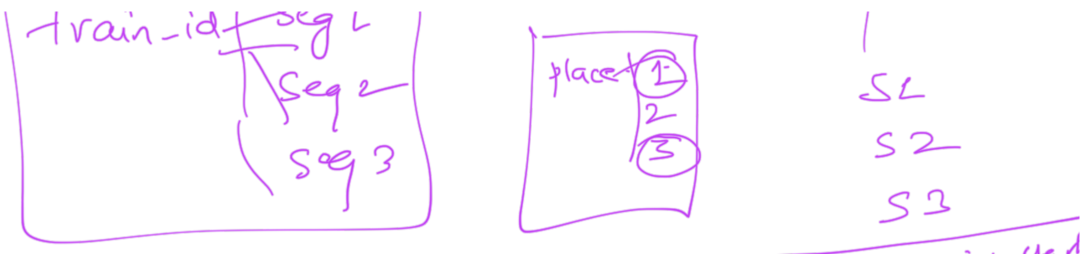






Seat	seqNo	UserId
		NULL

3A	S1	NULL
3A	S2	NULL
3A	S3	

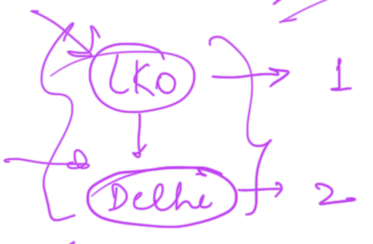


Train → all segments.

→ station → segment_id.

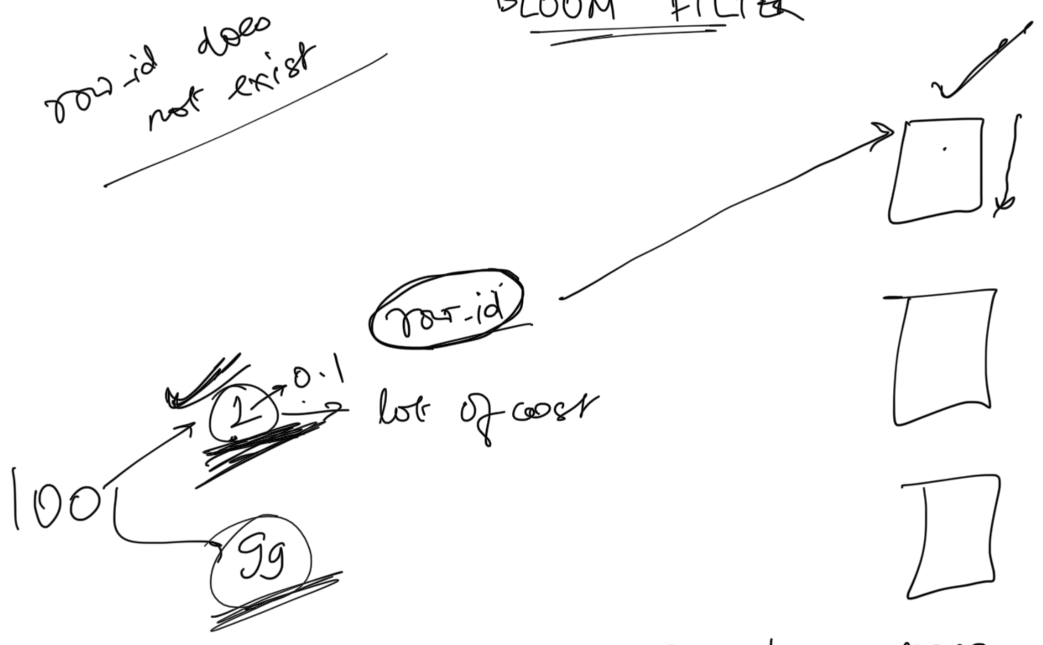
Seat → seat_no | seg_id

train_id + date → of origin sta
200 GB
↓
1.7B



S1 + S2

BLOOM FILTER

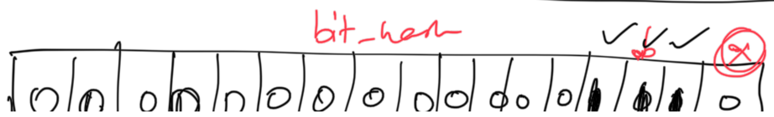


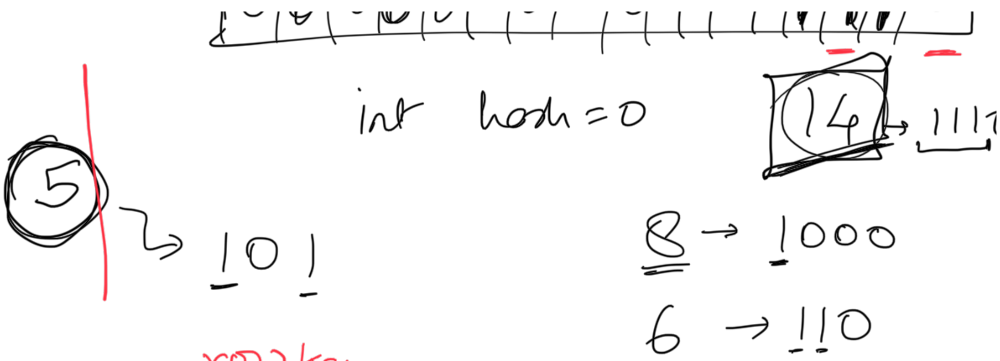
algorithm → uses very less space

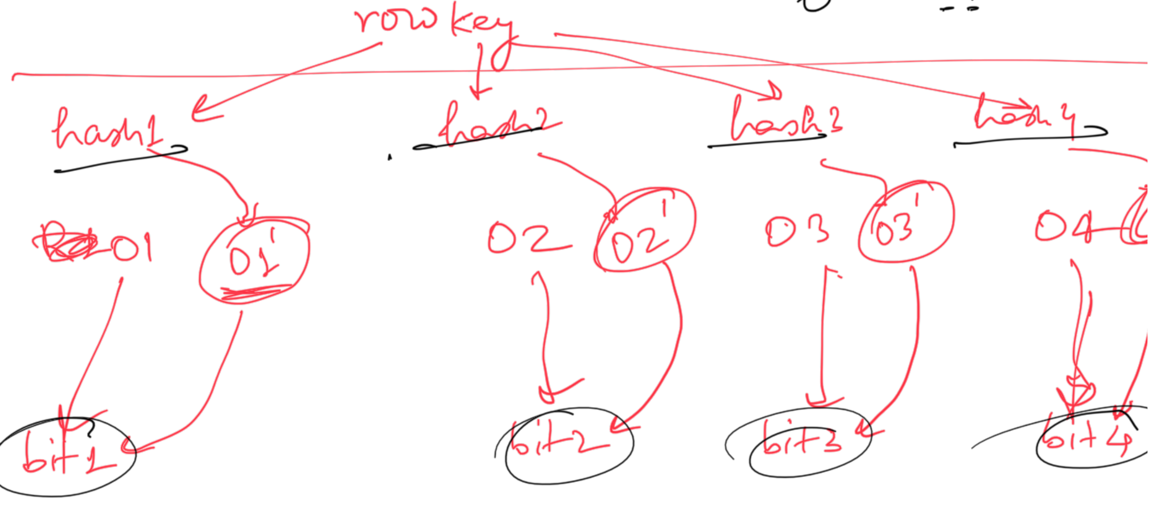
algorithm → key → YES → key still does not exi

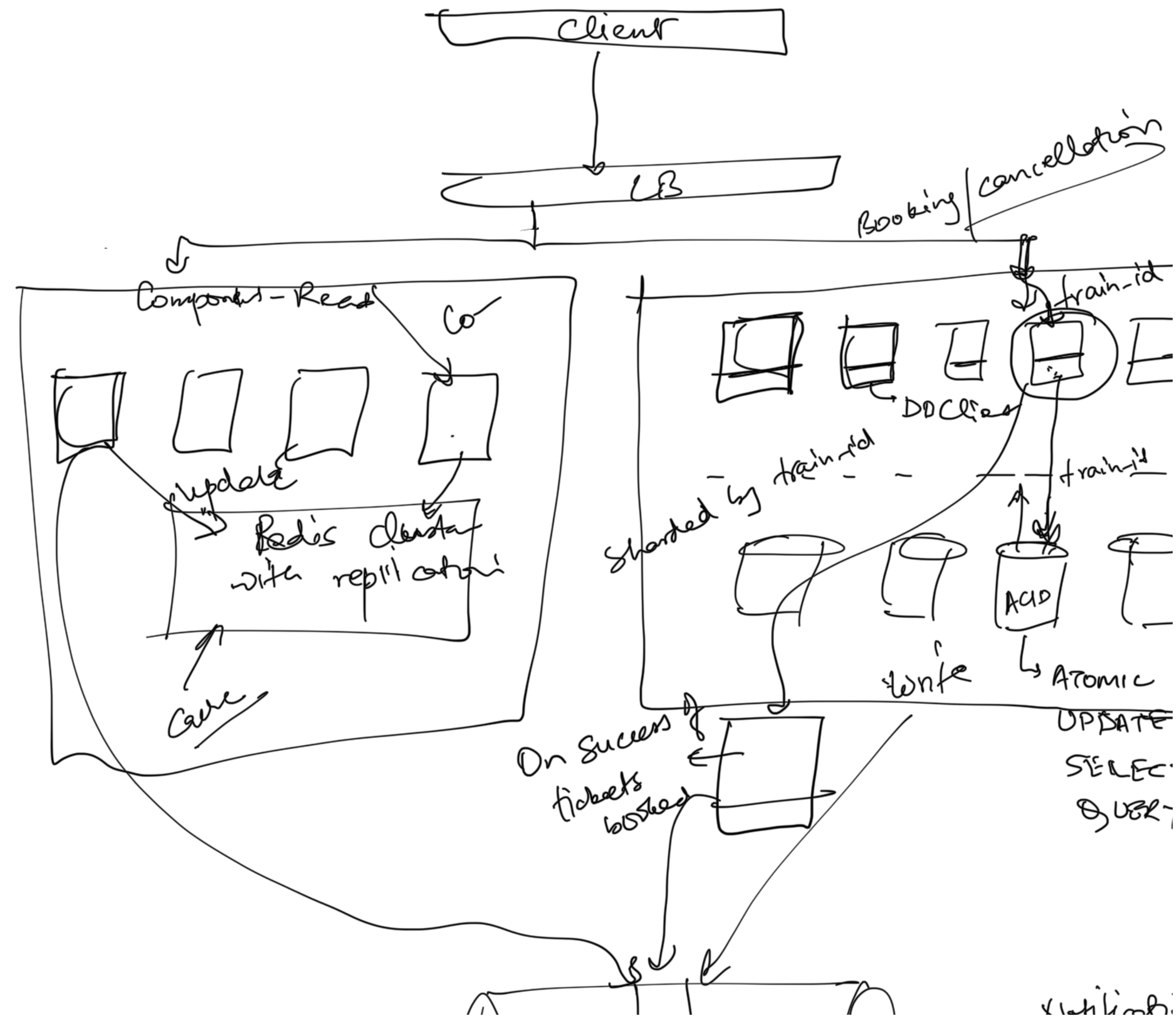
key → NO | 100% correct, relia

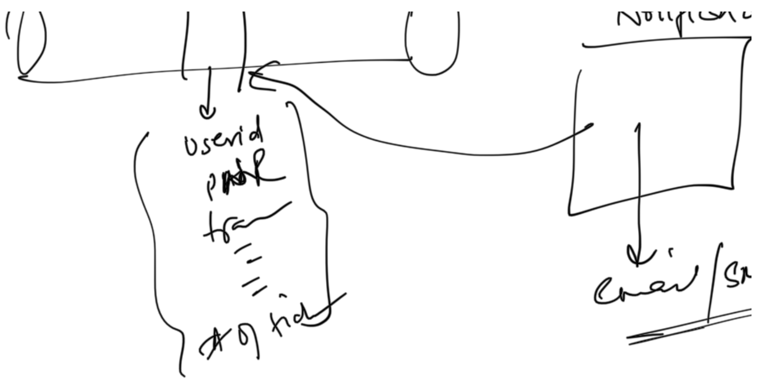
int →




int hash=0
5 → 101
14 → 111
8 → 1000
6 → 110


row key
hash1
hash2
hash3
hash4
01
01'
02
02'
03
03'
04
bit1
bit2
bit3
bit4


client
LB
Booking | cancellation
Component - Read
Co
update
Redis cluster with replication
Cache
train_id
DOCfiles
Sharded by train-id
train-id
ACID
ATOMIC
write
UPDATE
SELECT
QUERY
On success of tickets booked



Notificati
userid
pass
tran
# of tick
email / SM