DESIGN GOOGLE SEARCH TYPEAHEAD

~~mic~~ mci

- michail jackson
- michelle obama
- michael jordon

1. Functional requirement
   MVP (Min. Viable Prod
2. Estimation of scale
   - 1. Sharding
   - 2. Read vs write
   - X [3. QPS → 1 machine Q

   Queries per second
3. Design goals / non-fu req.
4. API design
   - system design

1 MVP

1. Given 3+ characters, give me top 5 suggestions of searc

Top = most freq. search

most trending.

mic

- ✓ - Personalisation → my searches, search from my region, search from my browse
- ✓ - Spelling mistakes → X
  - all suggestions will have pe typed text as strict prefix

Est. of scale

michael jackson [Enter]

michael jackson

1. Shard → YES
   - phrases → count
   - → new search term

6-8 billion searc day

10 billion se per day

30% searches are new

3 billion new phrases / day

*365*20

3B * 50 bytes

= *365*20

* 365 *

* 365 *

~~≃ 1PB~~

= 3 * 50 * 365 * 20

= 1000 TB ≃ 1PB

② Read heavy vs write heavy

Both read & write heavy.

① Read → "mic" → get Top5 s

② Write → any search → update count / fre

Design goals / Non Functional

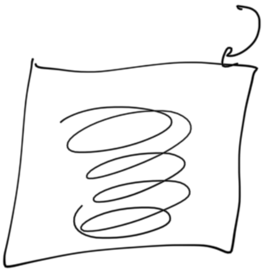
HA ✓

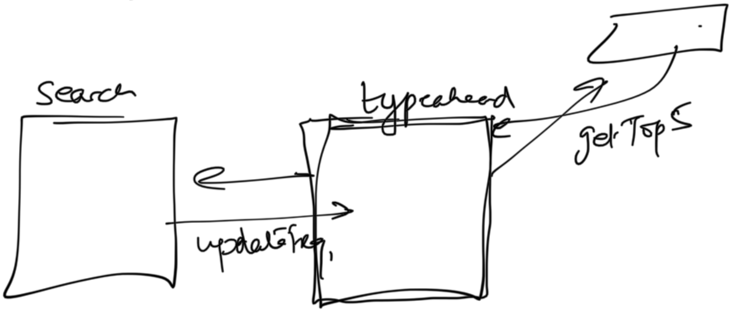
Latency of reads → Super low. ✓

API

① getTop5Suggestions(typed_str)

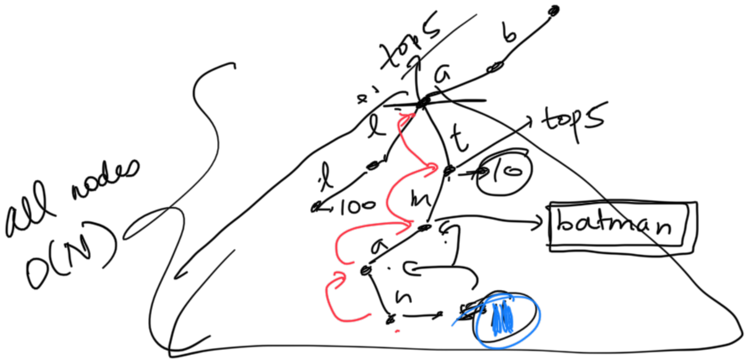
② updateFreq(search-term)





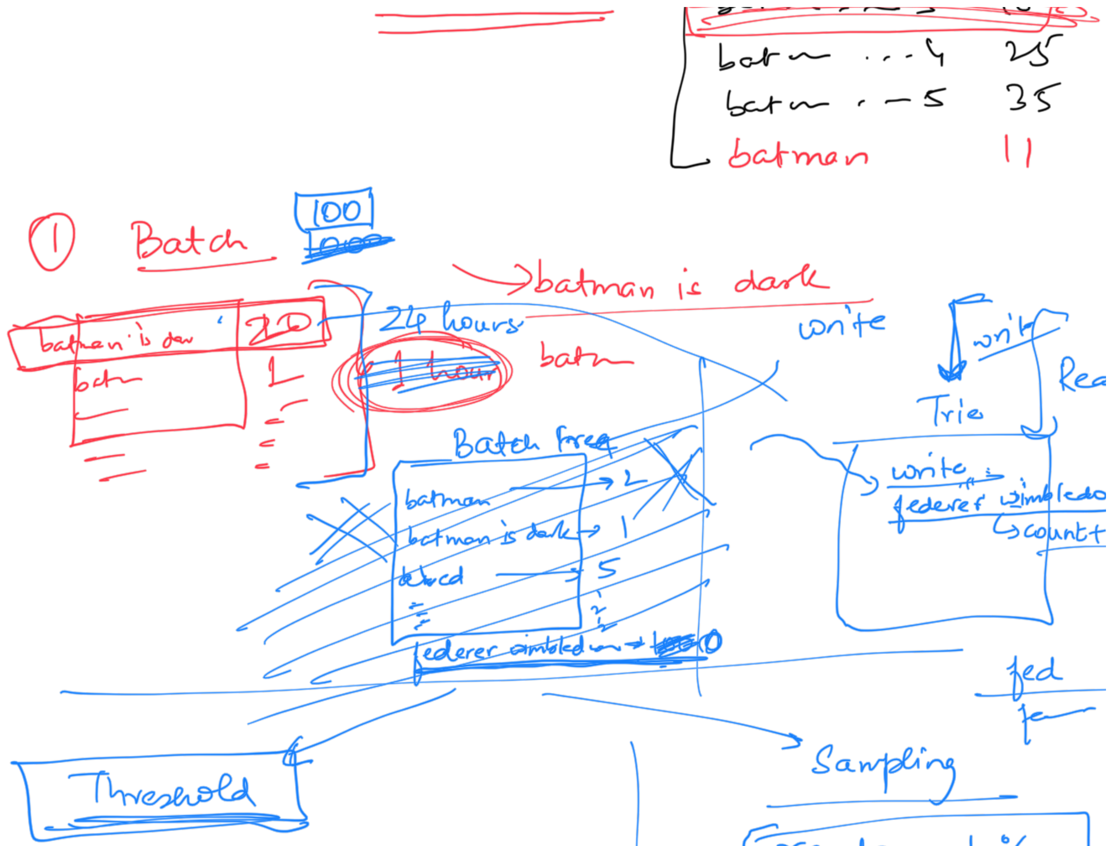
ba

```
node {
    int freq
    list<pair<string, i
}
```



batman is dark

batm ...	~~1~~	~~10~~ 20
batm ...	2	30
~~batm ...~~	~~?~~	~~10~~



random 1% searches

1% random writes

```
if(rand()%100 == 0){      0 - 99
    process the ...
} else {
    // ignore
}
```

~~2 Read vs write heavy~~

~~100~~

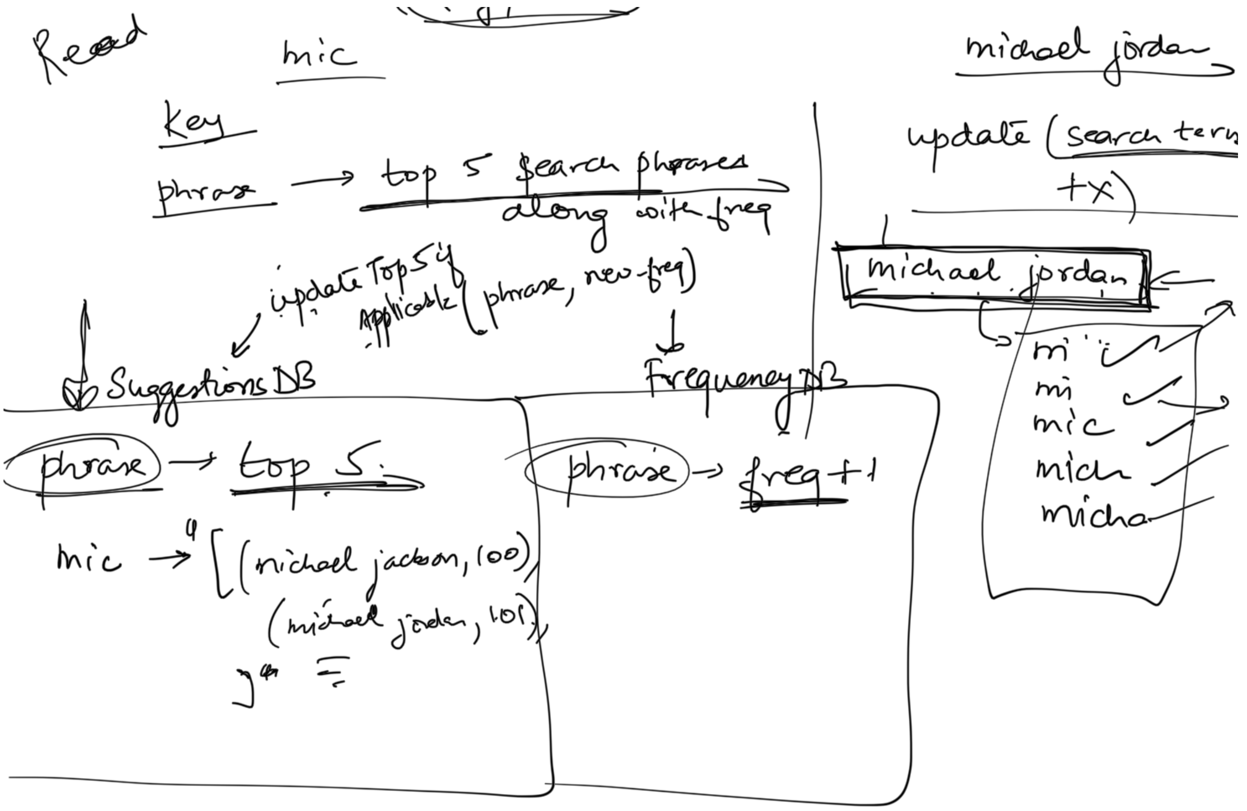
SAMPLING

```
updateFreq(search_term) {
    if(rand()%100 != 0) return; //99% of the ca...

    // logic of ~~write~~ updateFreq

}
```

(Key, value) . mic



```
for(i=0 ; i< str.length; i++)
    { str[0:i]
    }
```

```
getTop5Suggestions(phrase_prefix) {

  return SuggestionsDB.find(phase_prefix)

}

update(search_term, count) {

        cur_freq = FrequencyDB. ~~get~~find(search_term)
        new_freq = cur_freq + count
        frequencyDB.update(search_term, new_freq)

        for (i=~~3~~; i< search_term.length() ; i++) {
                     prefix = search_term[0: i]

              top5 = SuggestionsDB.find(prefix)
                  if (search_term in top5) {
                                       // update search term f
                                           in top5
                  } else if (new_freq > top5's least fre
```

consider loading sharding key

Key → value

// remove least freq term
and replace it with search
term, neo-

}

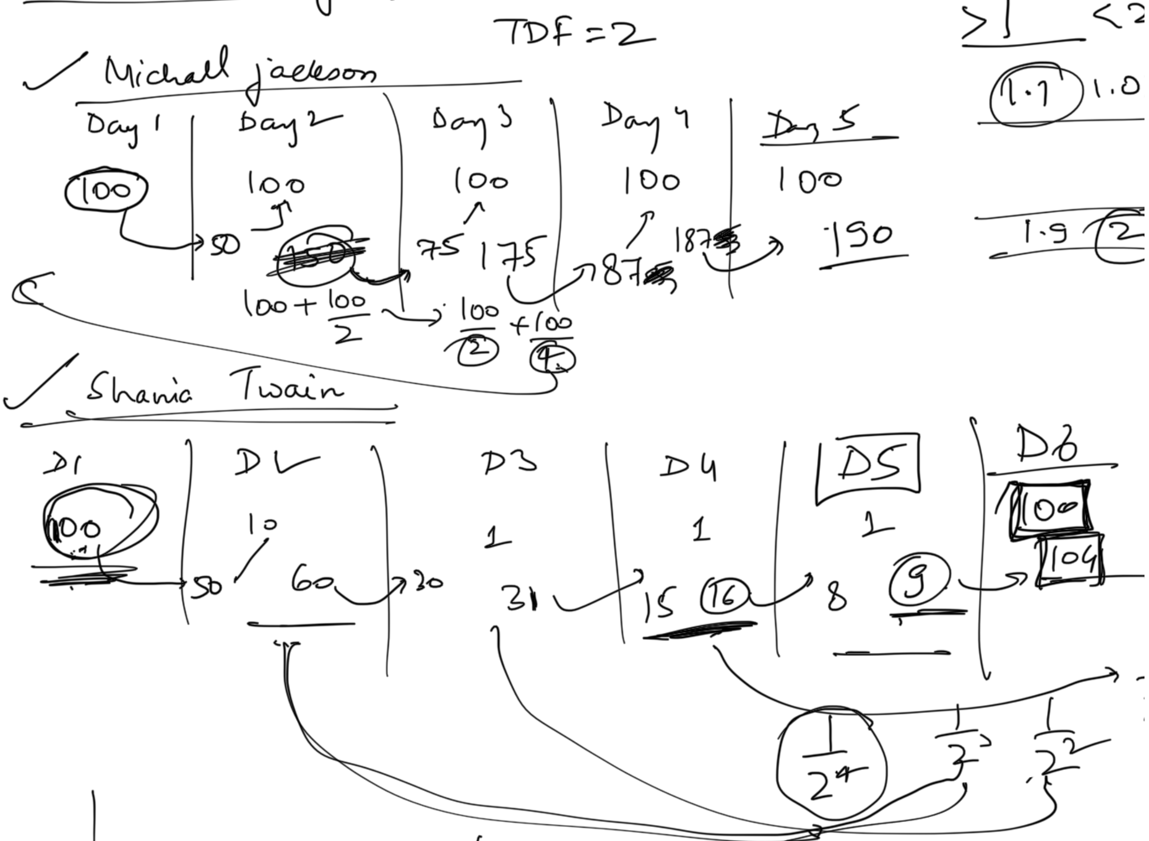
SuggestionsDB.update(prefix, top5)

}

---

michael jackson → 100,000,000

Time Decay factor



Count = count / TDF

Hashmap vs Trie

Serialise

HOD

TDF = 2

Anshuman

Day 1	Day 2	Day 3
240	240	$\boxed{360}$ 120+60 = 180

curr + $\frac{prev}{2}$

11:59

f = freq/2

12:05

An
↓
Anil

Anil

Day 1	Day 2	Day 3
240	240	$\boxed{260}$ 180

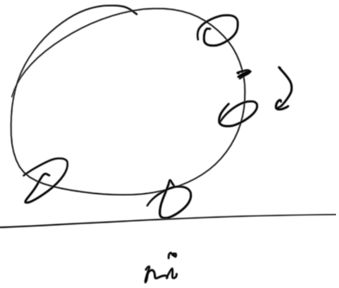
K, V

- supports sharding
- persistence
- HA
- high throughput
- ~~high~~ read heavy system

HC HA	read write heavy	KV doc Cf

key, value

Hash (key)

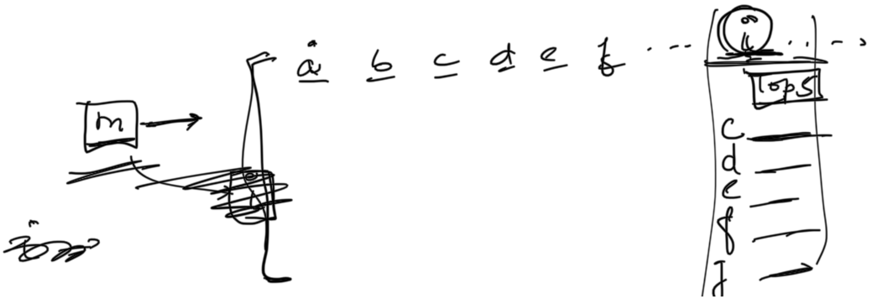


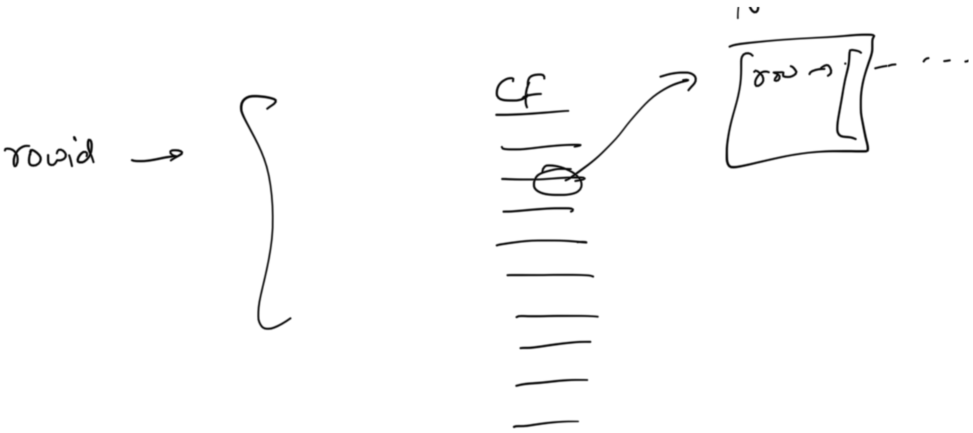
mic

26 words.

CF

mic




rowid
CF
row ->
...