DESIGN A URL-SHORTNER

clarifications questions -

1. Example of a URL shortner
2. what is the traffic volume
3. How long is the shortened URL?
4. what characters are allowed in the shortened URL? combination of numbers and characters.
5. Can shortened URLs be deleted (or) updated?

   for simplicity let's say NO

→ Basic use-cases of URL-SHORTNER

→ Given a long URL → returns a much shorter URL

→ URL redirecting; given a shorten URL → redirect to original URL

→ High availability, scalability and fault tolerance

Back of the envelope estimation

1 → 24
24 → 3600

→ Write operation → 100×10^6 URLs / day

→ write operations/sec = $\dfrac{100 \times 10^6}{24 / 3600}$ = 1160

→ Read operation → Assuming ratio of write to read is 10:1,

read/sec = 11.600

→ Assuming this URL shortening will run for 10 years, means

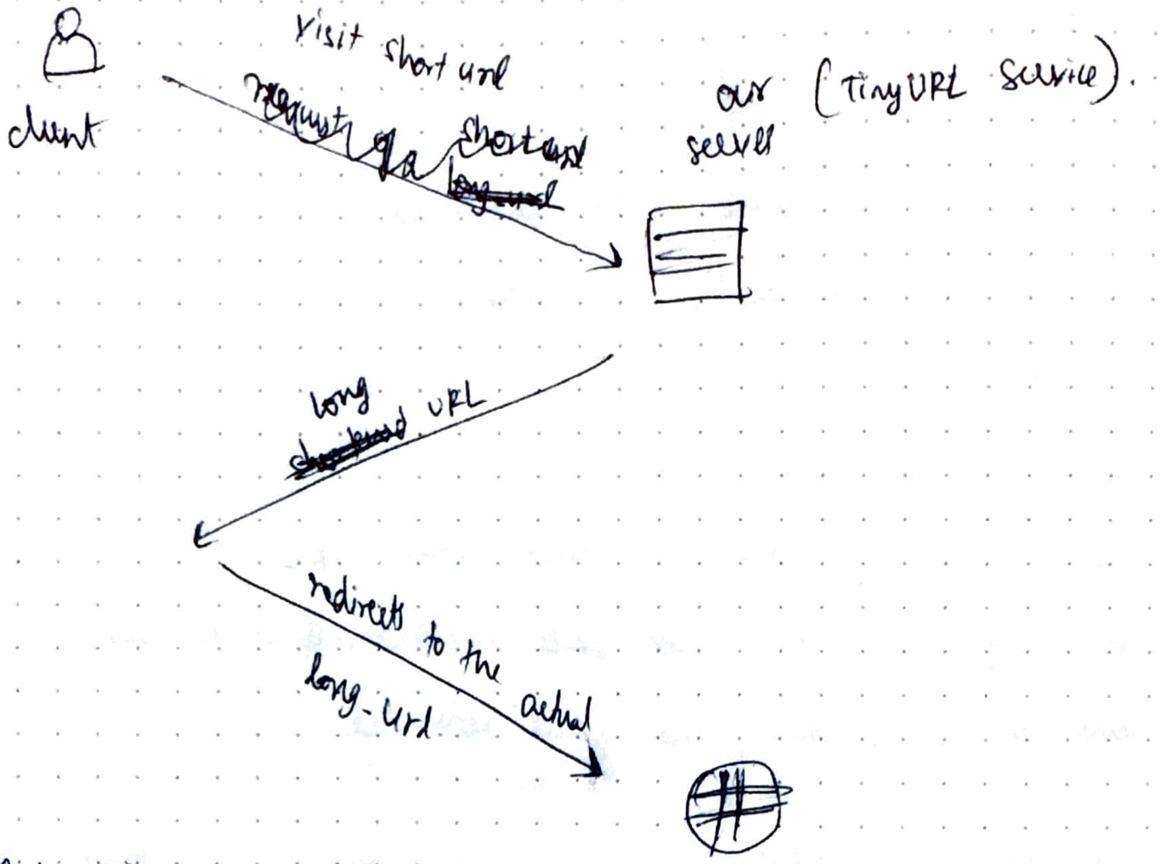
$\boxed{10^6 \times 100} \times 365 \times 10$ = 365 billion records.

100 milli.

→ Assume average URL length is 100

→ storage requirement for 10 years = 365 billion × 100 bytes × 10
= 365 TB

A typical scenario


client
visit short url
~~request~~ ~~for a~~ ~~shorten~~ ~~long url~~
our server
(TinyURL service).
long ~~shortened~~ URL
redirects to the actual long-url

Redirection in 2 ways :-

301 : → permanent redirect

302 : → Temporary redirect

Say if we want to save on the n/w bandwidth then we use 301 → permanent redirect.

So, we will cache the tinyURL, so everytime a request is made, it is fetched from there instead of then main service.

302 → Takes some time, but maybe used for some usecase
↳ long URL is temporarily moved to the long URL
↳ Can be useful for some metrics

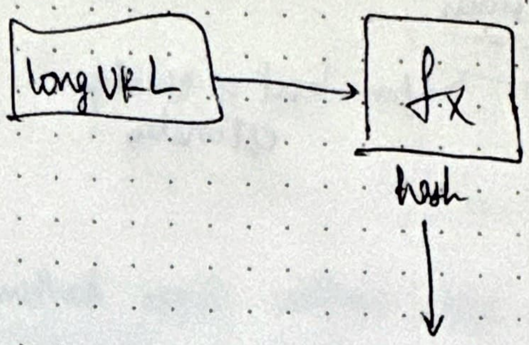
The most intuitive way to design a URL redirecting would be using hash table. Assuming the hash stores <shortURL, longURL> pairs.

Get longURL → hashTable.get(shortURL)

once we get the longURL, perform the re-direction

High Level View

Assume URL looks like this www.tinyurl.com/{hashvalue}. So, we must find a hash function fx that maps a long URL to the hashvalue



https://tinyurl.com/qtj5Opu

The hash function must satisfy the following requirements.

(i). Each longURL must be hashed to one hash value

(ii). Each hashvalue can be ~~to~~ mapped back to the longURL.

As hash tables are only in-memory so cannot be used for a distributed system, so we jump to databases.

and these are limited and expensive

PK
auto-increment

id	long url	shorturl

Hash Value

The hash value consists of characters from $\overset{10+26+26}{[0-9,\ a-z,\ A-Z]}$
62 possible characters. So, we need to select the smallest n such that $62^n \geq 365$ billion

from back of envelope estimates

So, we choose $n = 7$

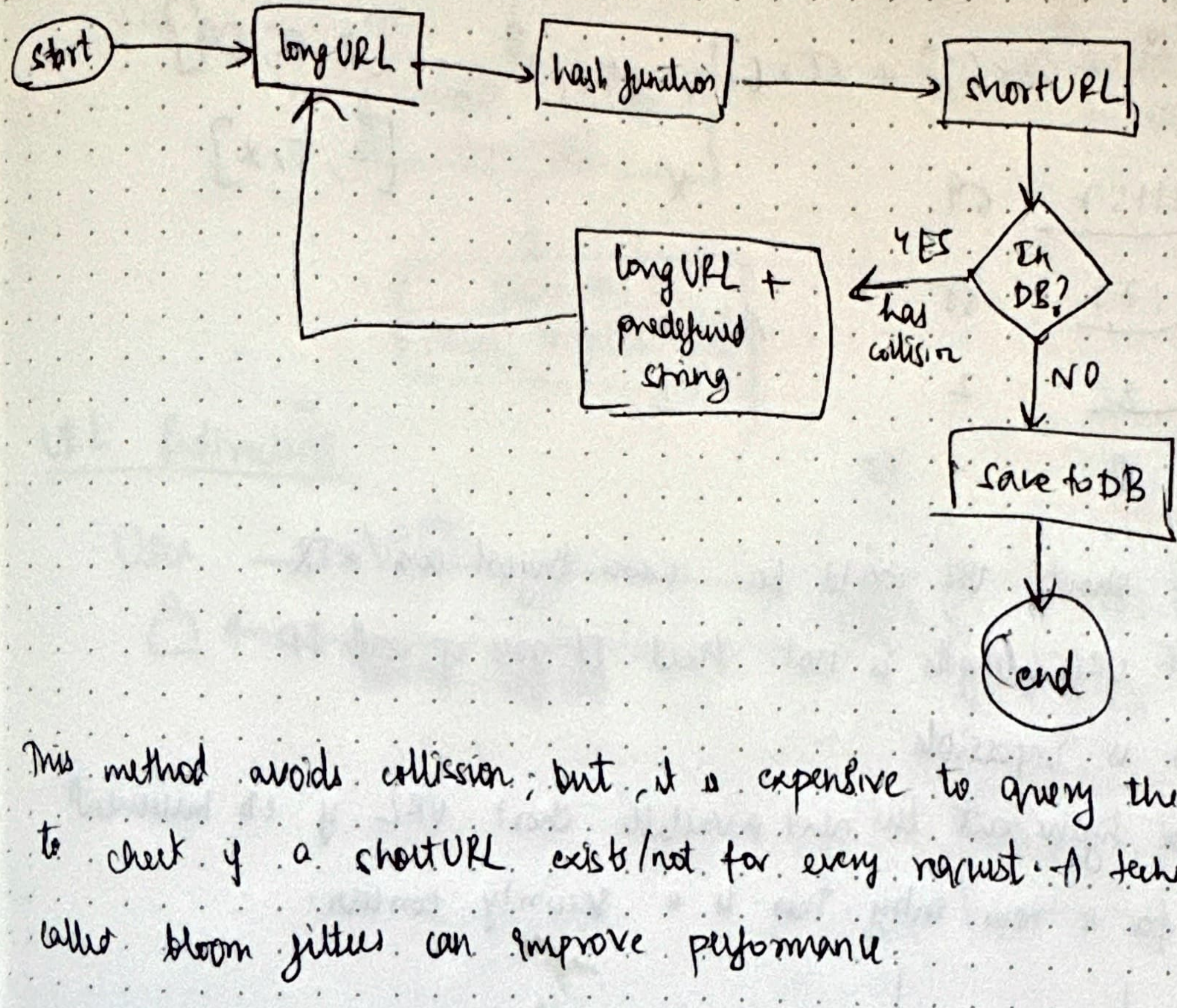
2 Types of Hash functions

(i) Hash + collission resolution

To shorten a long URL, we should implement a hash function that hashes a long URL to a 7-character string
Straightforward solution would be to use hash-functions like CRC32, MD5 (or) SHA1.

↳ these hash functions provide too long hash values. 7 7 characters -

↳ So, the first approach is to collect the first 7 characters of a hash value; but this can lead to collisions. So to resolve hash collisions we can recursively append a new predefined string until no more collision is discovered.



This method avoids collission; but it is expensive to query the database to check if a shortURL exists/not for every request. A technique called bloom filters can improve performance.

(ii) Base 62 conversion

Commonly used for URL shortners. Base conversions helps to convert the same number b/w its different representation systems. Base 62 is used as there are 62 possible characters

$11157_{10} \longrightarrow$ Base 62

$2 \times 62^2 + 55 \times 62^1 + 59 \times 62^0 \rightarrow [2, 55, 59]$

$[2, T, X]$

62	11157	59		X
62	179	55		T
62	2	2	↑	2
	0			

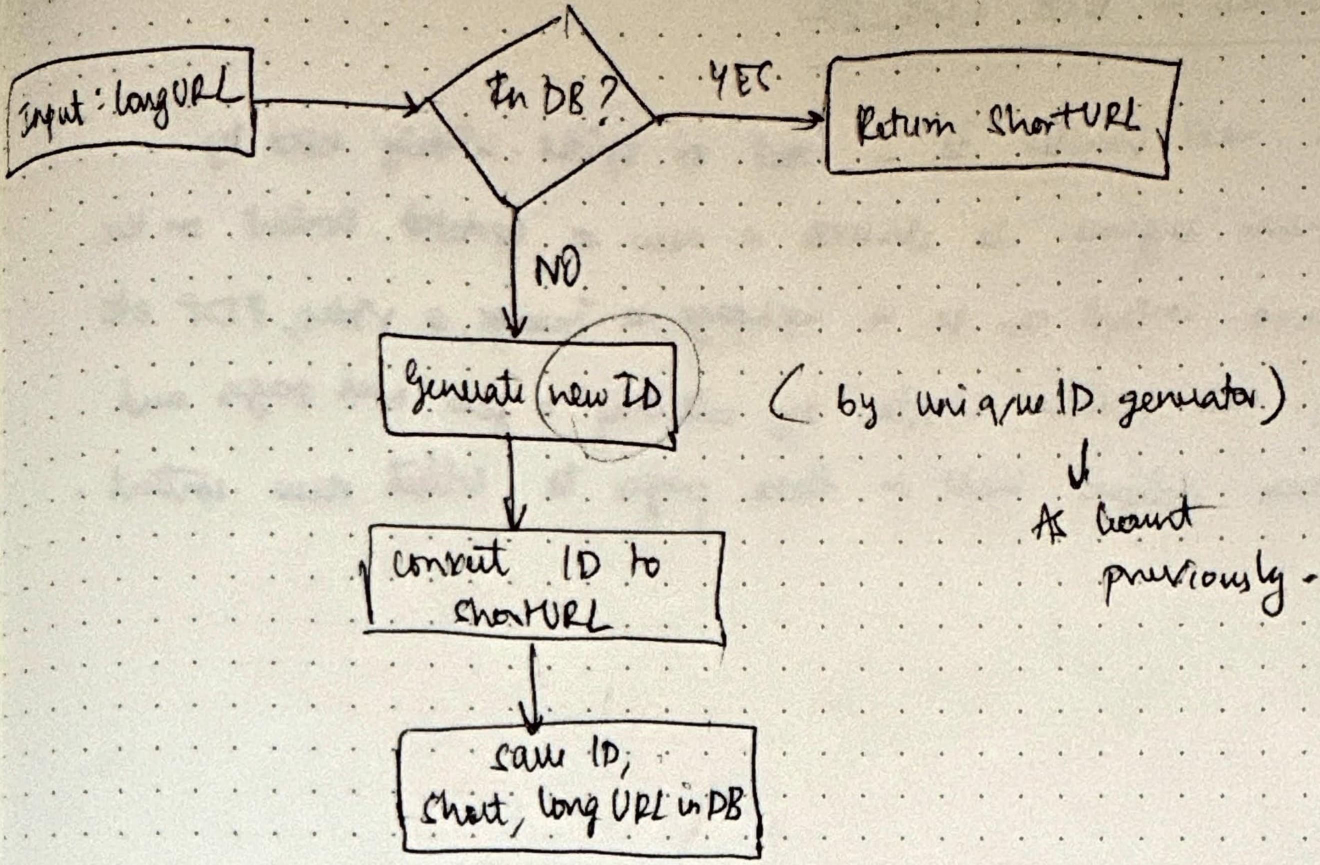
So, the short URL would be www. tinyurl. com/2TX

→ Here, short URL length is not fixed. It goes up with ID

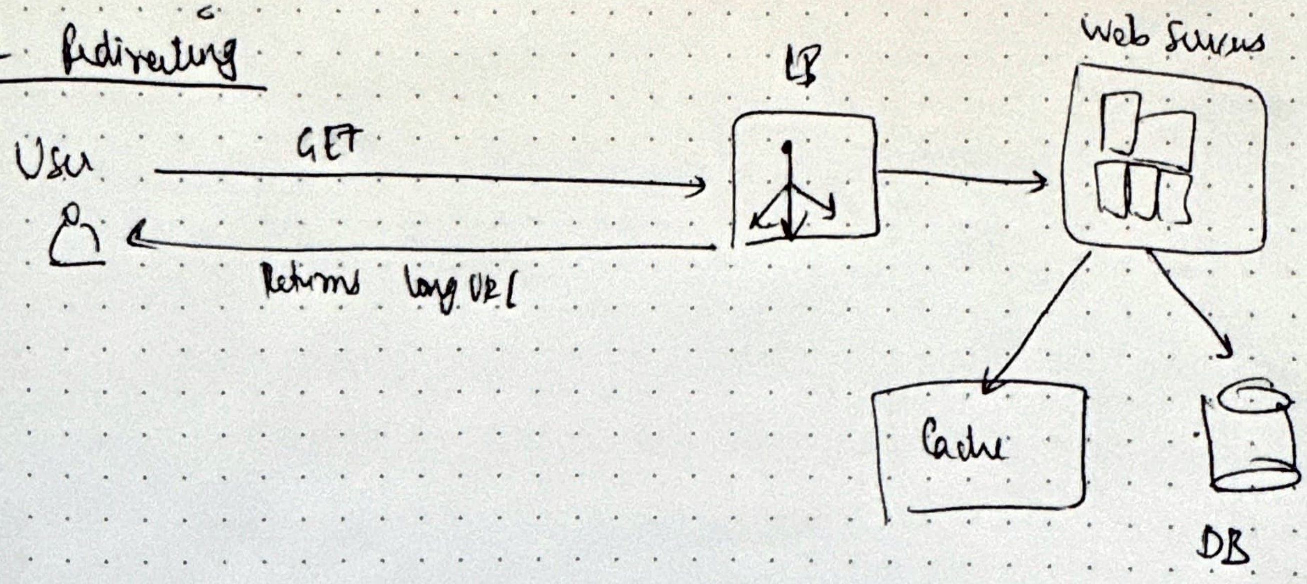
→ Collision is impossible

→ Easy to figure out the next available short URL if ID increments by 1 for a new entry. This is a security concern.

0 → 0	a → 10	A → 36
1 → 1	b → 11	B → 37
⋮		
9 → 9	z → 35	Z → 61



<u>URL Redirecting</u>


User
GET
Returns longURL
LB
Web Servers
Cache
DB