Step 3 - Design deep dive

Up until now, we have discussed the high-level design of URL shortening and URL redirecting. In this section, we dive deep into the following: data model, hash function, URL shortening and URL redirecting.

Data model

In the high-level design, everything is stored in a hash table. This is a good starting point; however, this approach is not feasible for real-world systems as memory resources are limited and expensive. A better option is to store <*shortURL, longURL*> mapping in a relational database. Figure 8-4 shows a simple database table design. The simplified version of the table contains 3 columns: *id, shortURL, longURL*.

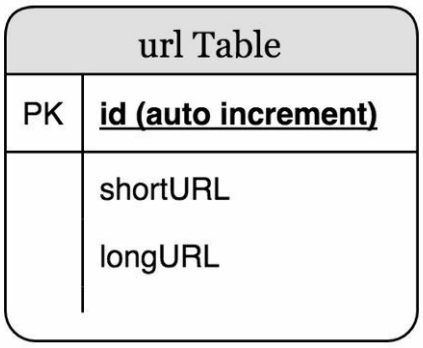

Figure 8-4

Hash function

Hash function is used to hash a long URL to a short URL, also known as *hashValue*.

Hash value length

The *hashValue* consists of characters from [0-9, a-z, A-Z], containing 10 + 26 + 26 = 62 possible characters. To figure out the length of *hashValue*, find the smallest *n* such that *62^n ≥ 365 billion*. The system must support up to 365 billion URLs based on the back of the envelope estimation. Table 8-1 shows the length of *hashValue* and the corresponding maximal number of URLs it can support.

N	Maximal number of URLs
1	62^1 = 62
2	62^2 = 3,844
3	62^3 = 238,328
4	62^ 4 = 14,776,336
5	62^5 = 916,132,832
6	62^6 = 56,800,235,584
7	62^7 = 3,521,614,606,208 = ~3.5 trillion

Table 8-1

When *n = 7, 62 ^ n = ~3.5 trillion*, 3.5 trillion is more than enough to hold 365 billion URLs, so the length of *hashValue* is 7.

We will explore two types of hash functions for a URL shortener. The first one is “hash + collision resolution”, and the second one is “base 62 conversion.” Let us look at them one by one.

Hash + collision resolution

To shorten a long URL, we should implement a hash function that hashes a long URL to a 7-character string. A straightforward solution is to use well-known hash functions like CRC32, MD5, or SHA-1. The following table compares the hash results after applying different hash functions on this URL: https://en.wikipedia.org/wiki/Systems_design.

Hash function	**Hash value (Hexadecimal)**
CRC32	5cb54054
MD5	5a62509a84df9ee03fe1230b9df8b84e
SHA-1	0eeae7916c06853901d9ccbefbfcaf4de57ed85b

Table 8-2

As shown in Table 8-2, even the shortest hash value (from CRC32) is too long (more than 7 characters). How can we make it shorter?

The first approach is to collect the first 7 characters of a hash value; however, this method can lead to hash collisions. To resolve hash collisions, we can recursively append a new predefined string until no more collision is discovered. This process is explained in Figure 8-

5.

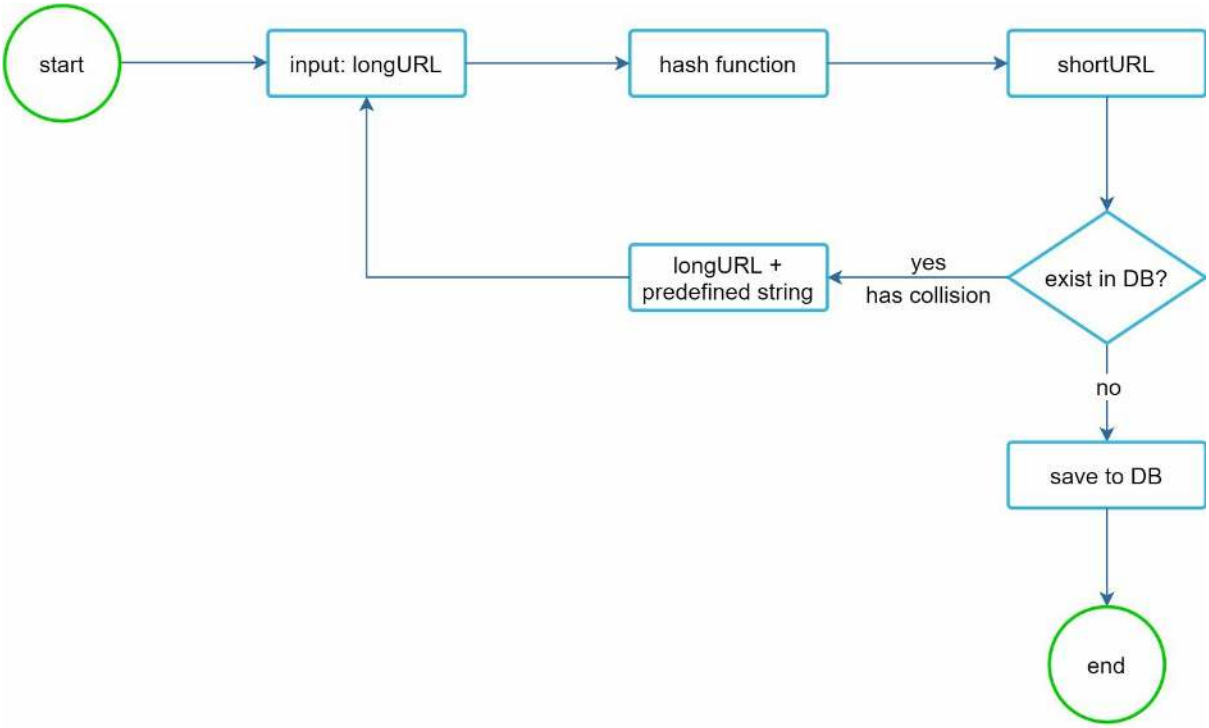

Figure 8-5

This method can eliminate collision; however, it is expensive to query the database to check if a shortURL exists for every request. A technique called bloom filters [2] can improve performance. A bloom filter is a space-efficient probabilistic technique to test if an element is a member of a set. Refer to the reference material [2] for more details.

Base 62 conversion

Base conversion is another approach commonly used for URL shorteners. Base conversion helps to convert the same number between its different number representation systems. Base 62 conversion is used as there are 62 possible characters for *hashValue*. Let us use an example to explain how the conversion works: convert 11157_{10} to base 62 representation (11157_{10} represents 11157 in a base 10 system).

- From its name, base 62 is a way of using 62 characters for encoding. The mappings are: *0-0, ..., 9-9, 10-a, 11-b, ..., 35-z, 36-A, ..., 61-Z, where* 'a' stands for 10, 'Z' stands for 61, etc.
- $11157_{10} = 2 \times 62^2 + 55 \times 62^1 + 59 \times 62^0 = [2, 55, 59] \rightarrow [2, T, X]$ in base 62 representation. Figure 8-6 shows the conversation process.

Figure 8-6

• Thus, the short URL is https://tinyurl.com /**2TX**

Comparison of the two approaches

Table 8-3 shows the differences of the two approaches.

Hash + collision resolution	**Base 62 conversion**
Fixed short URL length.	The short URL length is not fixed. It goes up with the ID.
It does not need a unique ID generator.	This option depends on a unique ID generator.
Collision is possible and must be resolved.	Collision is impossible because ID is unique.
It is impossible to figure out the next available short URL because it does not depend on ID.	It is easy to figure out the next available short URL if ID increments by 1 for a new entry. This can be a security concern.

Table 8-3

URL shortening deep dive

As one of the core pieces of the system, we want the URL shortening flow to be logically simple and functional. Base 62 conversion is used in our design. We build the following diagram (Figure 8-7) to demonstrate the flow.

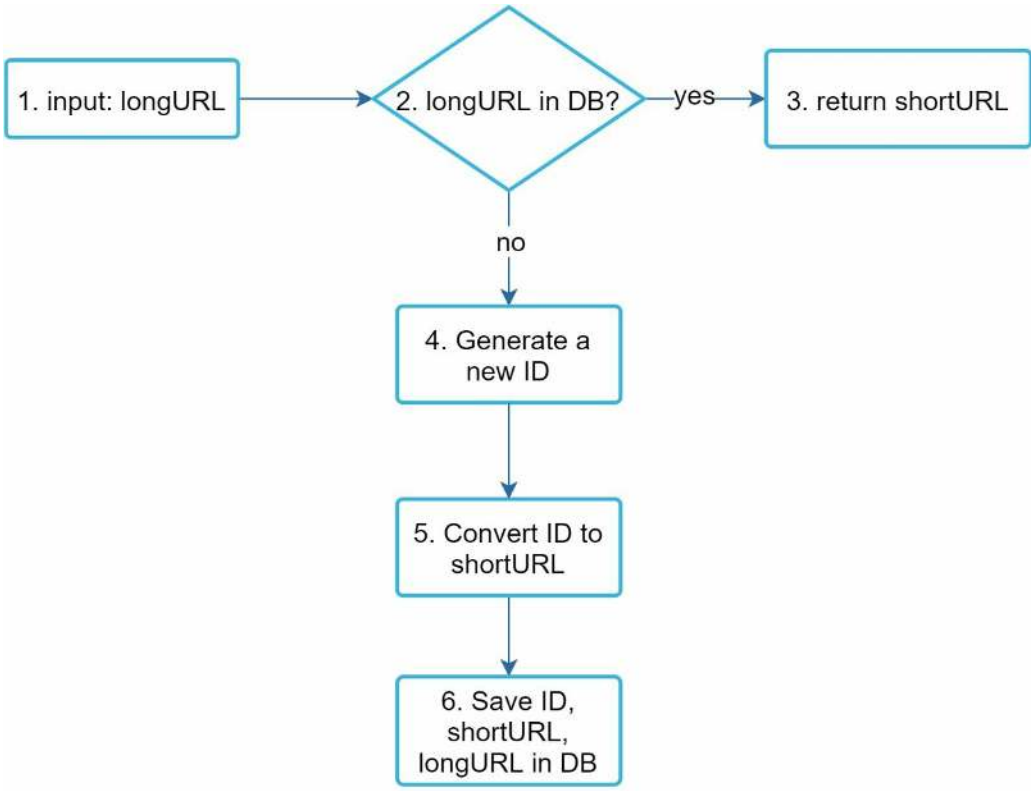

Figure 8-7

1. longURL is the input.
2. The system checks if the longURL is in the database.
3. If it is, it means the longURL was converted to shortURL before. In this case, fetch the shortURL from the database and return it to the client.
4. If not, the longURL is new. A new unique ID (primary key) Is generated by the unique ID generator.
5. Convert the ID to shortURL with base 62 conversion.
6. Create a new database row with the ID, shortURL, and longURL.

To make the flow easier to understand, let us look at a concrete example.

- Assuming the input longURL is: https://en.wikipedia.org/wiki/Systems_design
- Unique ID generator returns ID: 2009215674938.
- Convert the ID to shortURL using the base 62 conversion. ID (2009215674938) is converted to “zn9edcu”.
- Save ID, shortURL, and longURL to the database as shown in Table 8-4.

id	shortURL	longURL
2009215674938	zn9edcu	https://en.wikipedia.org/wiki/Systems_design

Table 8-4

The distributed unique ID generator is worth mentioning. Its primary function is to generate globally unique IDs, which are used for creating shortURLs. In a highly distributed

environment, implementing a unique ID generator is challenging. Luckily, we have already discussed a few solutions in “Chapter 7: Design A Unique ID Generator in Distributed Systems”. You can refer back to it to refresh your memory.

URL redirecting deep dive

Figure 8-8 shows the detailed design of the URL redirecting. As there are more reads than writes, <*shortURL, longURL*> mapping is stored in a cache to improve performance.

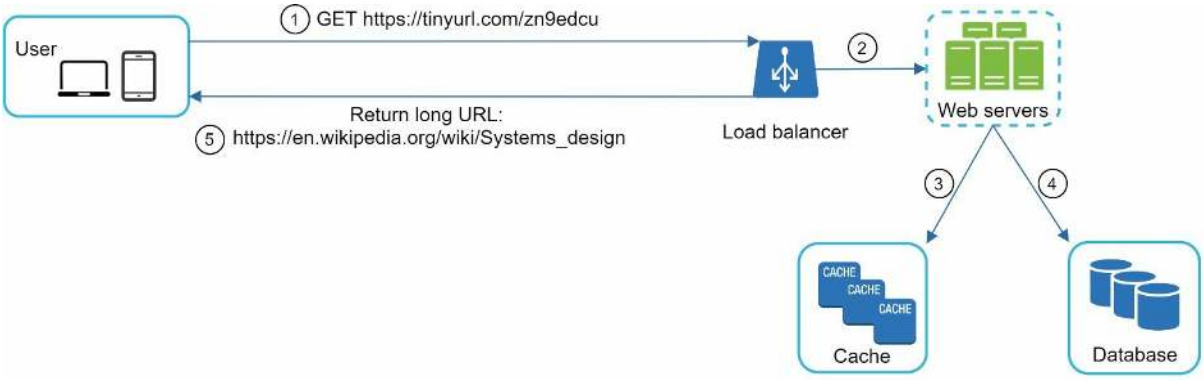

Figure 8-8

The flow of URL redirecting is summarized as follows:

1. A user clicks a short URL link: https://tinyurl.com/zn9edcu
2. The load balancer forwards the request to web servers.
3. If a shortURL is already in the cache, return the longURL directly.
4. If a shortURL is not in the cache, fetch the longURL from the database. If it is not in the database, it is likely a user entered an invalid shortURL.
5. The longURL is returned to the user.