8장 - URL 단축기 설계

1단계 : 문제 이해 및 설계 범위 확정

```
지원자: URL 단축기가 어떻게 동작해야 하는지 예제를 보여주실 수 있을까요? <

면접관: https://www.엄청긴-URL.com 이렇게 주어진다고 했을 때,
      이 서비스는 https://짧은-URL.com 과 같이 단축 URL을 결과로 제공
      짧은-URL로 접속했을 때 원래 URL로 갈 수 있어야 하죠.

지원자: 트래픽 규모는 어느 정도일까요?

면접관: 매일 1억(100million) 개의 단축 URL을 만들어 낼 수 있어야 합니다

지원자: 단축 URL의 길이는 어느 정도여야 하나요?

면접관: 짧으면 짧을수록 좋습니다.

지원자: 단축 URL에 포함될 문자에 제한이 있습니까?

면접관: 단축 URL에는 숫자(0부터 9까지)와 영문자(a부터 z, A부터 Z까지)만

지원자: 단축된 URL을 시스템에서 지우거나 갱신할 수 있습니까?

면접관: 시스템을 단순화하기 위해 삭제나 갱신은 할 수 없다고 가정합시다.
```

> 위 요구사항을 정리한 내용

1. URL 단축 : 주어진 긴 URL을 훨씬 짧게 줄인다.
2. URL 리다이렉션(redirection): 축약된 URI로 HTTP 요청이 오면 원래 URI로 안내
3. 높은 가용성과 규모 확장성, 그리고 장애 감내가 요구된다.

개략적 추정

- 쓰기 연산 : 매일 1억 개의 단축 URL 생성
- 초당 쓰기 연산 : 1억(100million) / 24 / 3600 = 1160
- 읽기 연산 : 읽기 연산과 쓰기 연산 비율은 10:1 이라고 가정. (초당 11,600회 발생 1160 * 10 = 11,600)
- URL 단축 서비스를 10년간 운영한다고 가정하면 1억(100million) * 365 * 10 = 3650(365billion) 개의 레코드를 보관해야 한다.
  - 축약 전 URL의 평균 길이는 100이라고 가정
  - 따라서 10년 동안 필요한 저장 용량은 3650억(365billion) * 100바이트 = 36.5TB이다.

이런 식으로 계산을 하고, 계산이 끝나면 면접관과 점검하고 합의한 뒤 다음을 진행하자.

2단계 : 개략적 설계안 제시 및 동의 구하기

API 엔드포인트

- 클라이언트는 REST 스타일로 서버가 제공하는 API 엔드포인트를 통해 서버와 통신
  - RESTful API에 대해 잘 모른다면 여기를 참고
- URL 단축기는 기본적으로 다음과 같은 두 개의 엔드포인트가 필요함
  1. **URL 단축용** 엔드포인트
     - 해당 엔드포인트에 단축할 URL을 인자로 실어 POST 요청

```
POST/api/v1/data/shorten
- 인자: {longUrl: longURLstring}
- 반환: 단축 URL
```

  2. **URL 리다이렉션용** 엔드포인트
     - 단축 URL에 대해서 HTTP 요청이 오면 원래 URL로 보내주기 위한 용도의 엔드포인트

```
GET/api/v1/shortUrl
- 반환: HTTP 리다이렉션 목적지가 될 원래 URL
```

URL 리다이렉션

다음 그림은 브라우저에 단축 URL을 입력하면 무슨 일이 생기는지 보여준다.

그림 8-1

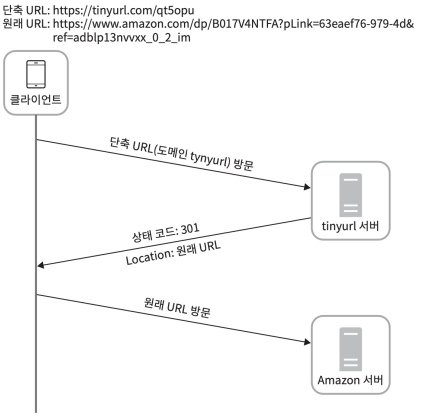

그림 8-2

- 단축 URL을 받은 서버는 해당 URL을 원래 URL로 바꾸어 `301 응답`의 Location 헤더에 넣어 반환
- 301과 302은 둘 다 리다이렉션에 대한 응답이지만 다음의 차이가 있음
  - **301 Permanently Moved**
    - 해당 URL에 대한 **HTTP 요청의 처리 책임이 영구적으로 Location 헤더에 반환된 URL로 이전** 됨
    - 영구적으로 이전되었으므로, 브라우저는 **이 응답을 캐시**한다.
      - 추후 **같은 단축 URL에 요청을 보낼** 필요가 있을 때 브라우저는 **캐시된 원래 URI로 요청**
  - **302 Found**
    - 주어진 URL로의 요청이 **일시적으로** Location 헤더가 지정하는 URL에 의해 처리되어야 하는 응답
    - 클라이언트의 요청은 언제나 단축 URL 서버에 먼저 보내진 후 원래 URL로 리다이렉션 되어야 함
- 위 두 방법은 다른 장단점을 갖고 있음
  - **서버의 부하를 줄이는** 것이 중요하다면 `301`을 사용 (첫 번째 요청만 단축 URL 서버로 전송되기 때문)
  - **트래픽 분석(analytics)**이 중요할 때는 `302` 사용 (클릭 발생률이나 발생 위치를 추적하는데 유리)

- **URL 리다이렉션을 구현하는 가장 직관적인 방법은 해시 테이블을 사용**
  - 해시 테이블에 `<단축 URL, 원래 URL>`의 쌍을 저장한다고 가정

```
원래 URL = hashTable.get(단축 URL)
301 또는 302 응답 Location 헤더에 원래 URL을 넣은 후 전송
```

URL 단축

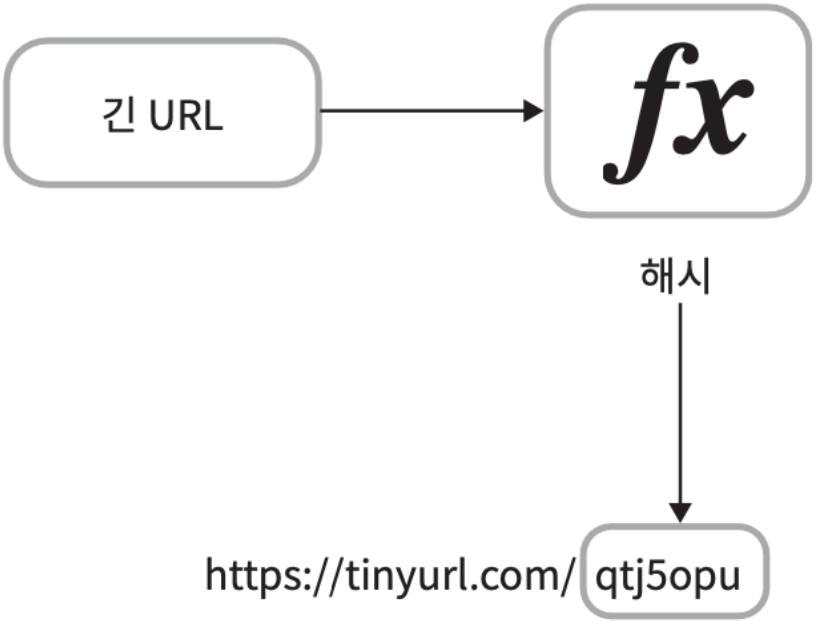

그림 8-3

단축 URI가 `www.abc.com/{hashValue}` 같은 형태라고 가정하고,

중요한 것은 긴 URL을 해시 값으로 대응시킬 해시 함수 fx를 찾는 것이다.

해시 함수가 지켜야 하는 요구사항

- 입력으로 주어지는 긴 URL이 다른 값이면 해시 값도 달라야 한다.
- 계산된 해시 값은 원래 입력으로 주어졌떤 긴 URL로 복원될 수 있어야 한다.

3단계 : 상세 설계

데이터 모델

앞서 개략적인 설계에서 **모든 것을 해시 테이블**에 뒀다.
이 방식은 초기 전략으로는 괜찮지만,
실제 시스템에서 사용하기엔 어렵다.

- 메모리는 유한, 비쌈

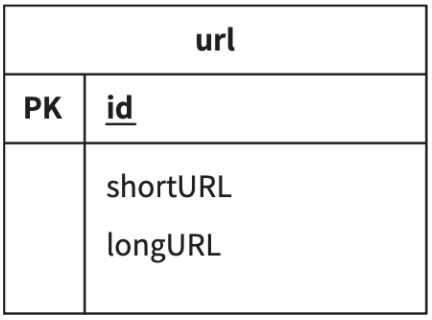

그림 8-4

세 가지 컬럼만 갖는다고 가정

따라서 위 메모리보다 더 좋은 방식은 `<단축 URL, 원래 URL>` 순서쌍을 RDB에 저장하는 형태

해시 함수

해시 함수 : 원래 URL을 단축 URL로 변환하는데 사용되는 함수, **계산된 결과값**을 `hashValue`라고 지칭

해시 값 길이

- `hashValue`는 `[0-9, a-z, A-Z]`의 문자들로 구성
  - 실제 사용할 수 있는 문자의 개수 10 + 26 + 26 = 62개
- `hashValue`의 길이를 정하기 위해서는 62^n ≥ 3650억 인 n의 최솟값을 찾아야 함
  - 개략적으로 계산했던 추정치에 따르면 이 시스템은 3650억 개의 URL을 만들 수 있어야 함

n	URL 개수
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	$62^7 = 3,521,614,606,208 = ~3.5$조(trillion)
8	$62^8 = 218,340,105,584,896$

표 8-1

- n = 7이면 3.5조 개의 URL을 만들 수 있고, 요구사항을 지치기 충분한 값이다.
- 따라서 hashValue의 길이는 7로 설정한다.

- 해시 함수 구현에 쓰일 기술로는 두 가지 방법을 사용

▼ **해시후 충돌 해소 방식**

- 긴 URL을 줄이려면, 원래 URL을 7즐자 문자열로 줄이는 해시 함수가 필요
  - 쉬운 방식으론 `CRC32`, `MD5`, `SHA-1`과 같이 잘 알려신 해시 함수를 이용

해시 함수	해시 결과 (16진수)
CRC32	5cb54054
MD5	5a62509a84df9ee03fe1230b9df8b84e
SHA-1	0eeae7916c06853901d9ccbefbfcaf4de57ed85b

표 8-2

CRC32, MD5, SHA-1 해시 함수를 사용한 예시

- 위 예시의 CRC32의 값도 7보다는 길어서 줄이는 방식
- 계산된 해시 값에서 **처음 7개 글자만 이용**
  - **해시 충돌 확률이 높아짐**
  - 충돌이 해소될 때까지 사전에 정한 문자열을 해시값에 덧붙여 해결

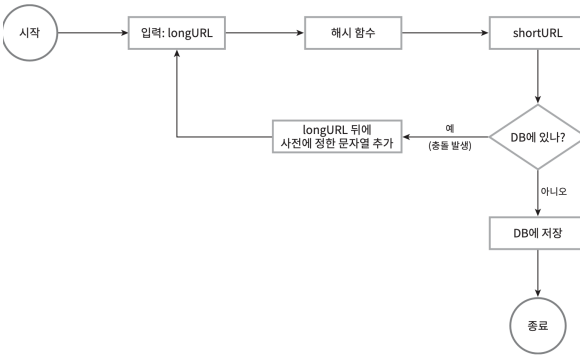

그림 8-5

- 이런 식으로 충돌을 해소할 수 있지만 오버헤드가 큼
- 데이터베이스 대신 **블룸 필터**를 사용하면 성능을 높일 수 있음
  - **블룸 필터** : 어떤 집합에 특정 원소가 있는지 검사할 수 있도록 검사
    (
    확률론에 기초한 공간 효율이 좋은 기술)

▼ **base-62 변환**

- **진법 변환(base conversion)**은 URL 단축기를 구현할 때 **흔히 사용**되는 접근법
- 이 기법은 수의 표현 방식이 다른 두 시스템이 같은 수를 공유해야 하는 경우에 유용
- 62진법을 쓰닌 이유는 `hashValue`에 사용할 수 있는 문자 개수가 62개이기 때문
- 11157을 62진수로 변환하면 다음과 같다.
  - `0은 0`, `9는 9`, `10은 a`, `11은 b`, … `35는 z`, `36은 A`, … `61은 Z`
  - 따라서 62진법에서 a는 1010을 나타내고, Z는 6110을 나타낸다.
  - 즉, **11157 = 2 * 62^2 + 55 * 62^1 + 59 * 62^0 = [2, 55, 59] ⇒ [2, T, X] ⇒ 2TX**

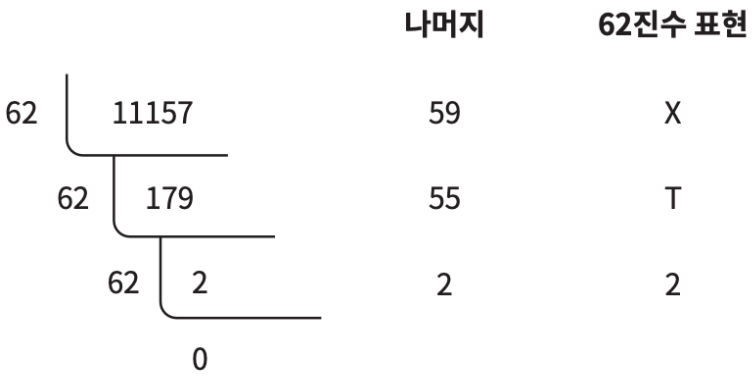

그림 8-6

- 따라서 단축 URL은 `www.abc.com/2TX` → 원본 URL `www.abc.com/11157`

두 접근법 비교

해시 후 충돌 해소 전략	base-62 변환
단축 URL의 길이가 고정됨	단축 URL의 길이가 가변적, ID 값이 커지면 같이 길어짐
유일성이 보장되는 ID 생성기가 필요치 않음	유일성 보장 ID 생성기가 필요
충돌이 가능해서 해소 전략이 필요	ID의 유일성이 보장된 후에야 적용 가능한 전략이라 **충돌은 아예 불가능**
ID로부터 단축 URL을 계산하는 방식이 아니라서 **다음에 쓸 수 있는 URL을 알아내는 것이 불가능**	ID가 1씩 증가하는 값이라고 가정하면 **다음에 쓸 수 있는 단축 URL이 무엇인지 쉽게 알아낼 수 있음** 이로 인해 **보안상 문제가 될 소지**가 있음

URL 단축 상세 설계

URL 단축기는 시스템의 핵심 컴포넌트이므로,
처리 흐름이 논리적으로는 단순해야 하고, 기능적으로는 언제나 동작하는 상태로 유지되어야 한다.

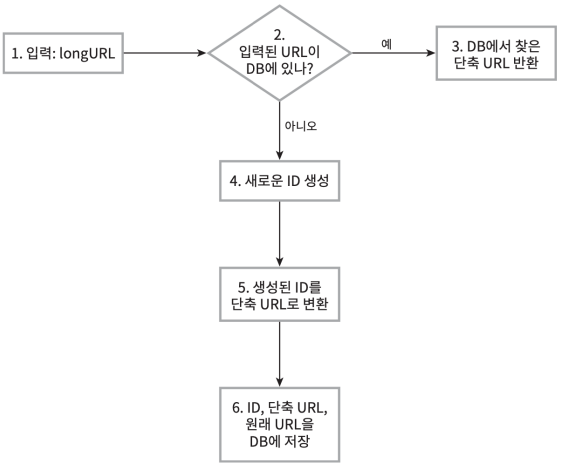

그림 8-7

1. 입력으로 긴 URL을 받는다.
2. 데이터베이스에 해당 URL이 있는지 검사한다.
3. 데이터베이스에 있다면 해당 URL에 대한 단축 URL을 만든 적이 있음
   따라서 데이터베이스에서 해당 단축 URL을 가져와 클라이언트에게 반환
4. 데이터베이스에 없는 경우 해당 URL은 새로 접수된 것 유일한 ID를 생성해야 됨
   해당 ID는 데이터베이스의 기본 키로 사용
5. 62진법 변환을 적용, ID를 단축 URL로 만듦
6. ID, 단축 URL, 원래 URL로 새 데이터베이스 레코드를 만든 후 단축 URL을 클라이언트에 전달

실전 예제

- 입력된 URL이 `https://www.abc.com`이라고 가정
- 이 URL에 대해 ID 생성기가 반환한 ID는 `2009215674938`이라고 가정
- 위 ID를 62진수로 변환하면 `zn9edcu`가 된다.
- 아래 표와 같은 새로운 데이터베이스 레코드를 만든다.

ID	shortURL	longURL
`2009215674938`	`zn9edcu`	`https://www.abc.com`

URL 리다이렉션 상세 설계

쓰기보다 읽기를 더 자주 하는 시스템이라, `<단축 URL, 원래 URL>`의 쌍을 캐시에 저장하여 성능을 높일 수 있다.

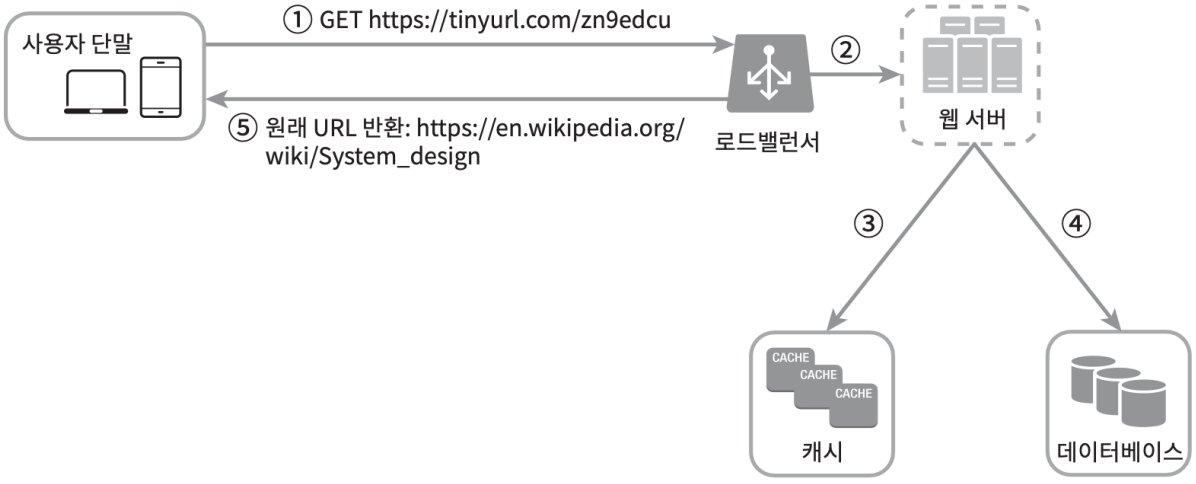

그림 8-8

> 로드밸런서의 동작 흐름은 다음과 같이 요약할 수 있음

1. 사용자가 단축 URL을 클릭
2. 로드밸런서가 해당 클릭으로 발생한 요청을 웹 서버에 전달
3. 단축 URL이 이미 캐시에 있는 경우 원래 URL을 바로 꺼내 클라이언트에게 전달
4. 캐시에 해당 단축 URL이 없는 경우에는 데이터베이스에서 꺼낸다.
   데이터베이스에 없다면 사용자가 잘못된 단축 URL을 입력한 경우
5. 데이터베이스에서 꺼낸 URL을 캐시에 넣은 후 사용자에게 반환

4단계 : 마무리

추가로 고려해볼 만한 것

- **처리율 제한 장치(rate limiter)**
  - 엄청난 양의 URL 단축 요청이 밀려들 경우 무력회될 수 있다는 잠재적 보안 결함을 갖고 있음
  - 처리율 제한 장치를 통해 IP 주소를 비롯한 필터링 규칙들을 이용해 요청을 걸러낼 수 있음

- **웹 서버의 규모 확장**
  - 이 설계에 포함된 웹 계층은 무상태(stateless) 계층이므로, 웹 서버를 증설하거나 삭제할 수 있음
- **데이터베이스의 규모 확장**
  - 데이터베이스를 다중화하거나 샤딩(sharding)하여 규모 확장성을 달성할 수 있음
- **데이터 분석 솔루션(analytics)**
  - URL 단축기에 데이터 분석 솔루션을 통합해 두면 어떤 링크를 얼마나 많은 사용자가 클릭했는지
    언제 주로 클릭했는지 등 중요한 정보를 알아낼 수 있음
- **가용성, 데이터 일관성, 안정성**
  - 대규모 시스템이 성공적으로 운영되기 위해서는 반드시 갖추어야 할 속성들
  - 이 부분은 1장을 복습하자.

Chapter 8: DESIGN A URL SHORTENER

1. A RESTful Tutorial
2. Bloom filter