8️⃣

231008 8. URL 단축기 설계

8. URL 단축기 설계

▼ 1. 문제 이해 및 설계 범위 확정

시스템 설계 면접 문제는 의도적으로 어떤 정해진 결말을 갖지 않도록 만들어짐

질문을 통해 모호함을 줄이고 요구사항을 알아내야 함

요구하는 시스템의 기본적 기능

1. URL 단축: 주어진 긴 URL을 훨씬 짧게 줄인다
2. URL 리디렉션: 축약된 URL로 HTTP 요청이 오면 원래 URL로 안내
3. 높은 가용성과 확장성, 장애 감내가 요구됨

개략적 추정

- 쓰기 연산
  - 매일 1억 개의 단축 URL 생성
- 초당 쓰기 연산
  - 1억(100million) / 24 / 3600 = 1160
- 읽기 연산
  - 읽기 연산과 쓰기 연산 비율이 10:1일 경우, 읽기 연산은 초당 11600회 발생
- URL 단축 서비스를 10년간 운영한다고 가정했을 때, 1억 * 365 * 10 = 3650억 개의 레코드를 보관해야 함
- 축약 전 URL의 평균 길이는 100이라고 가정했을 때, 10년 동안 필요한 저장 용량은 3650억 * 100바이트 = 36.5TB

▼ 2. 개략적 설계안 제시 및 동의 구하기

▼ API 엔드포인트

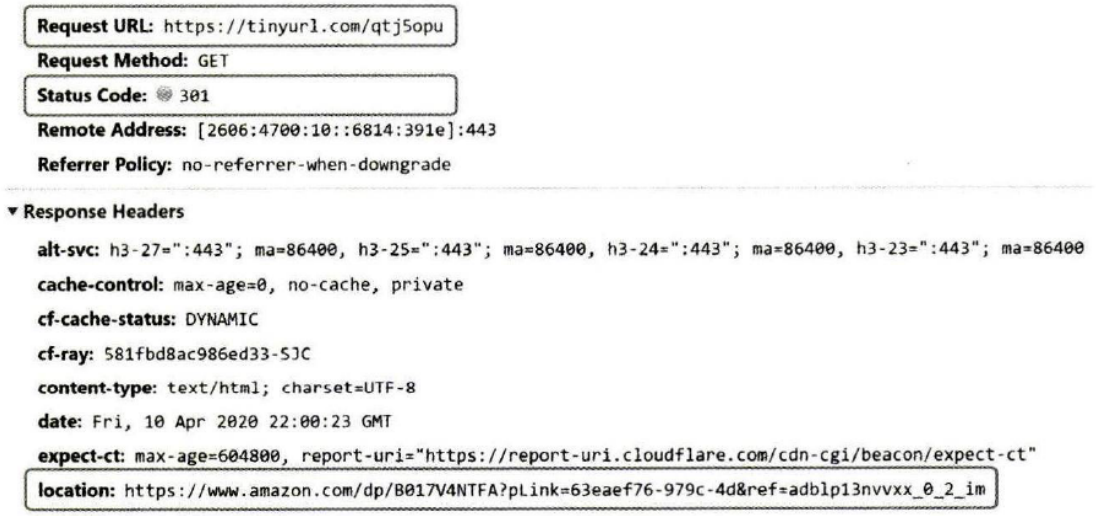
클라이언트는 서버가 제공하는 API 엔드포인트를 통해 서버와 통신함

REST 스타일로 설계

URL 단축키는 두 개의 엔드포인트가 필요함

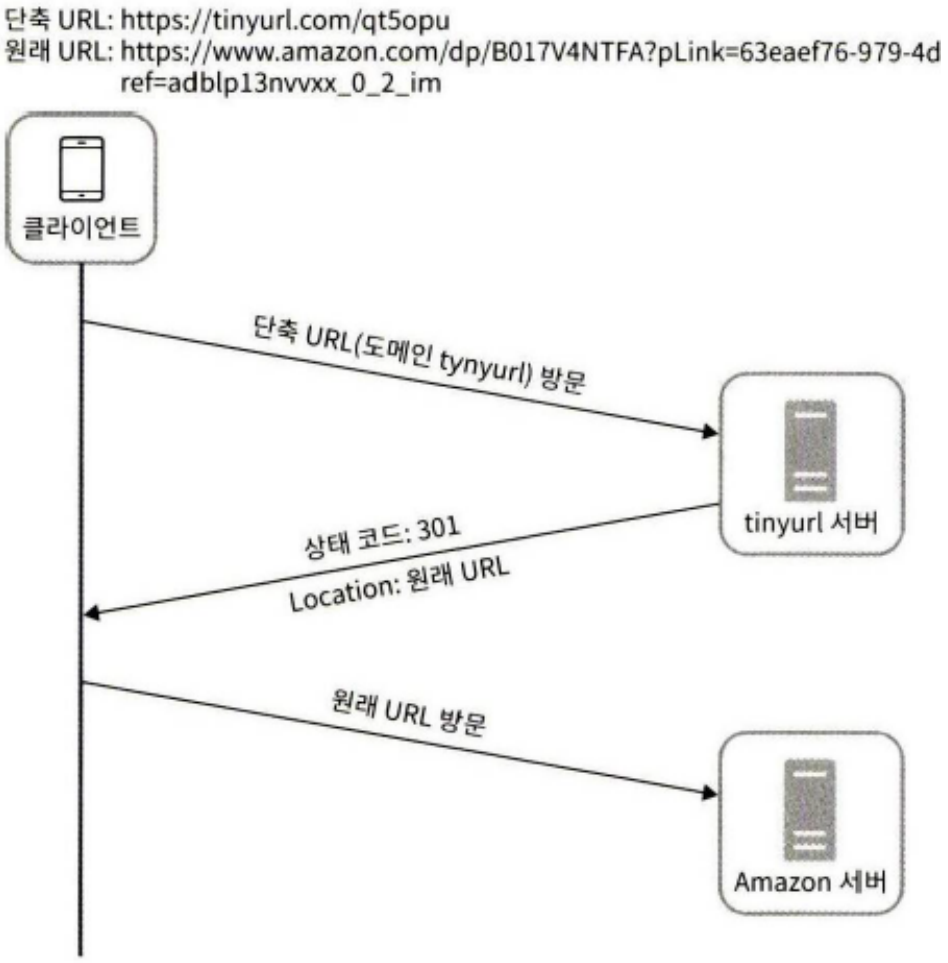
1. URL 단축용 엔드포인트
    - 새 단축 URL을 생성하고자 하는 클라이언트는 이 엔드포인트에 단축할 URL을 인자로 실어서 POST 요청을 보내야 함
    - `POST /api/v1/data/shorten`
        - 인자: {longUrl: longURLstring}
        - 반환: 단축 URL
2. URL 리다이렉션용 엔드포인트
    - 단축 URL에 대해서 HTTP 요청이 오면 원래 URL로 보내주기 위한 용도
    - `GET /api/v1/shortUrl`
        - 반환: HTTP 리다이렉션 목적지가 될 원래 URL

▼ URL 리디렉션



브라우저에 단축 URL을 입력하면 생기는 일

단축 URL을 받은 서버는 그 URL을 원래 URL로 바꾸어서 301 응답의 Location 헤더에 넣어 반환



- 301 Permenently Moved
  - 해당 URL에 대한 HTTP의 요청의 처리 책임이 영구적으로 Location 헤더에 반환된 URL로 이전되었다는 응답
  - 영구적으로 이전되었기 때문에 브라우저는 이 응답을 캐싱
  - 추후 같은 단축 URL에 요청을 보낼 필요가 있을 때 브라우저는 캐시된 원래 URL로 요청을 보냄
  - 서버 부하를 줄일 수 있음
- 302 Found
  - 주어진 URL로의 요청이 일시적으로 Location 헤더가 지정하는 URL에 의해 처리되어야 한다는 응답이다
  - 클라이언트의 요청은 언제나 단축 URL 서버에 먼저 보내진 후에 원래 URL로 리다이렉션 되어야 함
  - 트래픽 분석이 중요할 때 사용

URL 리다이렉션을 구현하는 가장 직관적인 방법은 **해시 테이블**을 사용하는 것

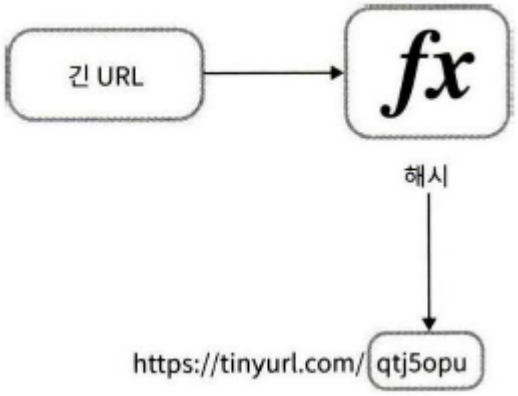
해시 테이블에 <단축 URL, 원래 URL>의 쌍을 저장한다고 가정

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

▼ URL 단축

단축 URL이 www.tinyurl/{hashValue} 같은 형태라고 가정

중요한 것은 긴 **URL을 이 해시 값으로 대응시킬 해시 함수 fx를 찾는 일**



요구사항

1. 입력으로 주어지는 긴 URL이 다른 값이면 해시 값도 달라야 함
2. 계산된 해시 값은 원래 입력으로 주어졌던 긴 URL로 복원될 수 있어야 함

▼ 3. 상세 설계

▼ 데이터 모델

모든 것을 해시 테이블에 두는 것은 초기 전략으로 괜찮지만 실제 시스템에 쓰기는 곤란

메모리는 유한한 데다 비쌈

더 나은 방법은 **<단축 URL, 원래 URL>의 순서쌍을 관계형 데이터베이스에 저장**하는 것

▼ 해시 함수

해시 함수는 원래 URL을 단축 URL로 변환하는 데 쓰임

편의상 해시 함수가 계산하는 단축 URL 값을 hashValue라고 지칭

해시 값 길이

hashValue는 [0-9, a-z, A-Z]의 문자들로 구성됨

사용할 수 있는 문자의 개수는 62(10 + 26 + 26)개

hashValue의 길이를 정하기 위해서는 3650억인 n의 최솟값을 찾아야 함

개략적으로 계산했던 추정치에 따르면 이 시스템은 3650억 개의 URL을 만들어 낼 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	$62^7 = 3,521,614,606,208 = \sim 3.5$조(trillion)
8	$62^8 = 218,340,105,584,896$

hashValue의 길이와 해시 함수가 만들 수 있는 URL 개수 사이의 관계

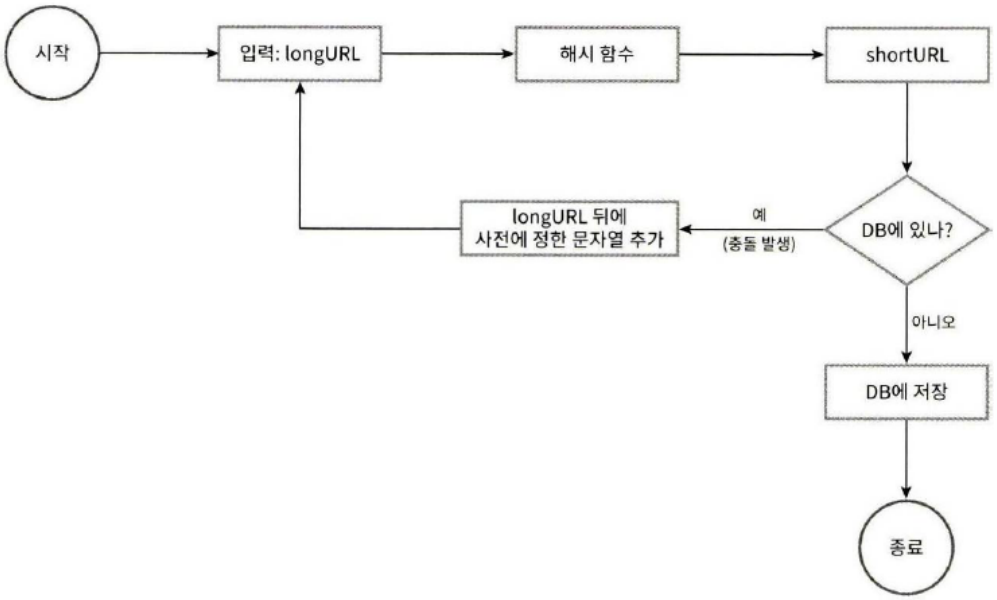
n이 7이면 3.5조 개의 URL을 만들 수 있음

요구사항을 만족시키기 충분한 값이기에 hashValue의 길이는 7로 가정

해시 함수 구현에 쓰이는 두 가지 기술

- 해시 후 충돌 해소
  - 긴 URL을 줄이려면 원래 URL을 7글자 문자열로 줄이는 해시 함수가 필요
  - 손쉬운 방법은 CRC32, MD5, SHA-1 같이 잘 알려진 해시 함수를 이용하는 것이지만 가장 짧은 변환 값도 7개 글자보다 길다는 문제가 있음
  - 이에 대한 첫 번째 해결 방안은 계산된 해시 값에서 처음 7개 글자만 이용하는 것
    - 결과가 서로 충돌할 확률이 높아짐

- 충돌이 실제로 발생했을 때는 충돌이 해소될 때까지 사전에 정한 문자열을 해시값에 덧붙임
- 충돌은 해소할 수 있지만 단축 URL을 생성할 때 한 번 이상 데이터베이스 질의를 해야 하기 때문에 오버헤드가 큼



  - 데이터베이스 대신 블룸 필터를 사용하면 성능을 높일 수 있음

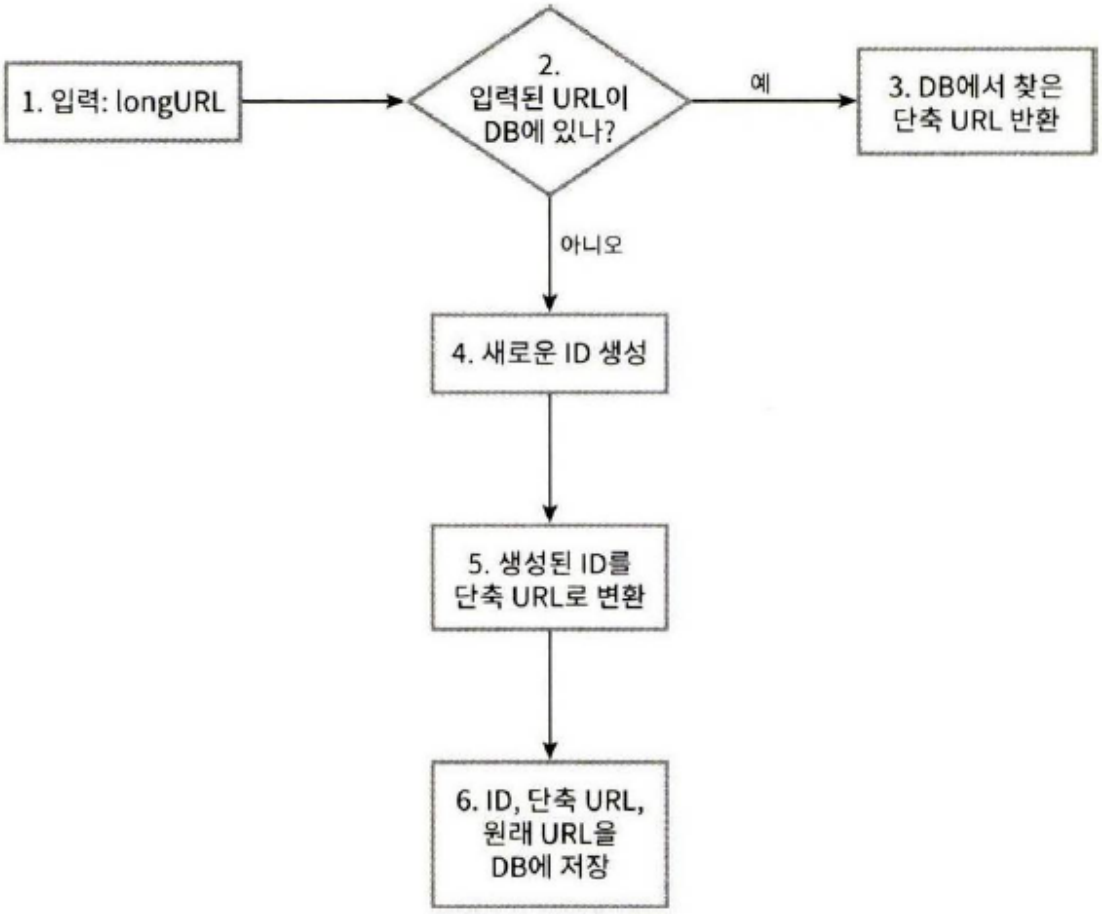
- base-62 변환
  - 진법 변환은 URL 단축키를 구현할 때 흔히 사용되는 접근법 중 하나
  - 수의 표현 방식이 다른 두 시스템이 같은 수를 공유해야 하는 경우에 유용
  - 62진법을 쓰는 이유는 hashValue에 사용할 수 있는 문자의 개수가 62개이기 때문
- 두 접근법 비교

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 있어서 보안상 문제가 될 소지가 있음

▼ URL 단축기 상세 설계

URL 단축키는 시스템의 핵심 컴포넌트

처리 흐름이 논리적으로 단순해야 하고 기능적으로는 언제나 동작하는 상태로 유지되어야 함



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

1. 입력으로 긴 URL 받기
2. 데이터베이스에 해당 URL이 있는지 검사
3. 데이터베이스에 있다면 해당 URL에 대한 단축 URL을 가져와서 클라이언트에 반환
4. 없다면 데이터베이스의 기본 키로 사용될 유일한 ID 생성
5. 62진법 변환을 적용하여 ID를 단축 URL로 변환

6. ID, 단축 URL, 원래 URL로 새 데이터베이스 레코드를 만든 후 단축 URL을 클라이언트에 반환

ID 생성기의 주된 용도는 단축 URL을 만들 때 사용할 ID를 만드는 것

ID는 전역적 유일성이 보장되어야 함

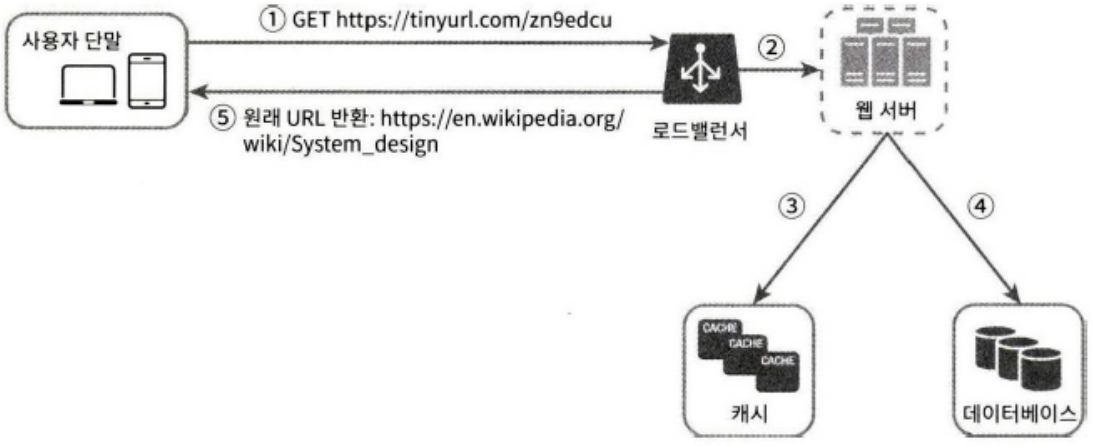
고도로 분산된 환경에서 이런 생성기를 만드는 것은 무척 어려운 일

▼ URL 리디렉션 상세 설계

URL 리디렉션 메커니즘의 상세한 설계

쓰기보다 읽기를 더 자주 하는 시스템

<단축 URL, 원래 URL>의 쌍을 캐시에 저장하여 성능을 높임



1. 사용자가 단축 URL을 클릭
2. 로드밸런서가 해당 클릭으로 발생한 요청을 웹 서버에 전달
3. 단축 URL이 이미 캐시에 있으면 원래 URL을 바로 꺼내서 클라이언트에게 전달
4. 없다면 데이터베이스에서 꺼내기
5. 데이터베이스에 없다면 아마 사용자가 잘못된 단축 URL을 입력한 경우
6. 데이터베이스에서 꺼낸 URL을 캐시에 넣은 후 사용자에게 반환

▼ 4. 마무리

URL 단축기를 설계하면서 추가적으로 고민해 볼 수 있는 사항

- 처리율 제한 장치(rate limiter)

    - 엄청난 양의 URL 단축 요청이 있을 경우 무력화될 수 있다는 잠재적 보안 결함을 갖고 있음
    - 처리율 제한 장치를 두면 IP 주소를 비롯한 필터링 규칙들을 이용해 요청을 걸러낼 수 있음

- 웹 서버의 규모 확장
    - 본 설계에 포함된 웹 계층은 무상태 계층이기 때문에 웹 서버를 자유롭게 증설하거나 삭제할 수 있음
- 데이터베이스의 규모 확장
    - 데이터베이스를 다중화하거나 샤딩하여 규모 확장성을 달성할 수 있음
- 데이터 분석 솔루션(analytics)
    - 성공적인 비즈니스를 위해서는 데이터가 중요
    - 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 속성

▼ 토론

URL을 단축할 때 파라미터를 함께 저장하는 것이 좋을지 아니면 제외하고 저장하는 것이 좋을지

파라미터를 함께 저장하는 경우

- **추적 가능성**: URL에 포함된 파라미터를 함께 저장하면, 클릭 데이터를 분석하거나 특정 파라미터 값을 추적하는 데 도움이 됩니다. 이는 마케팅 캠페인 분석이나 사용자 동작 이해에 유용할 수 있습니다.
- **사용자 정의**: 특정 사용자가 URL을 단축하고 파라미터를 지정할 수 있는 기능을 제공할 수 있습니다. 이를 통해 사용자는 보다 맞춤형된 단축 URL을 생성할 수 있습니다.
- **유용성**: 일부 사용 사례에서는 파라미터가 중요한 정보를 전달하는 데 필요할 수 있으며, 이 정보를 잃지 않도록 보존하는 것이 유용할 수 있습니다.

파라미터를 제외하고 저장하는 경우

- **간결성**: 파라미터 없이 URL을 저장하면, URL이 더 간결하고 깔끔하게 보일 수 있습니다. 이는 URL을 공유할 때 사용자 경험을 개선할 수 있습니다.
- **보안**: 파라미터 없이 저장하면 URL에 민감한 정보가 포함되지 않으므로, 보안상의 우려를 줄일 수 있습니다.
- **단순화**: 파라미터를 다루지 않는 것은 URL 단축 서비스의 복잡성을 줄일 수 있으며, 서비스의 유지 및 관리를 단순화할 수 있습니다.

어떤 방식을 선택할지는 서비스의 목표와 사용자 요구사항에 따라 다를 것입니다. 예를 들어, 마케팅 캠페인 추적을 목표로 하는 경우 파라미터 저장이 유용할 수 있으나, 개인 정보 보호가 중요한 경우 파라미터를 저장하지 않는 것이 바람직할 수 있습니다. 따라서 파라미터 저장 여부를 결정할 때 사용자 프라이버시와 데이터 보안에 주의를 기울이는 것이 중요합니다.