3️⃣

# 231029 13장. 검색어 자동완성 시스템

검색창에 단어를 입력하면 입력 중인 글자에 맞는 검색어가 자동으로 표시되는 것

## ▼ 문제 이해 및 설계 범위 확정

**요구사항**

- 빠른 응답 속도
  - 사용자가 검색어를 입력함에 따라 충분히 빨리 표시되어야 함
  - 페북 기준 100밀리초 이내
- 연관성
  - 사용자가 입력한 단어와 연관되어야 함
- 정렬
  - 시스템의 계산 결과는 인기도 등의 순위 모델에 의해 정렬되어야 함
- 규모 확장성
  - 많은 트래픽을 감당할 수 있도록 확장 가능해야 함
- 고가용성
  - 시스템의 일보에 장애가 발생하거나, 느려지거나, 예상치 못한 네트워크 문제가 발생하더라도 시스템은 계속 사용 가능해야 함

**개략적 규모 추정**

- 일간 능동 사용자(DAU)는 천만 명
- 평균적으로 한 사용자는 매일 10건의 검색을 수행
- 질의할 때마다 평균 20바이트의 데이터 입력
  - 문자 인코딩 방법은 ASCII(1문자 = 1바이트)

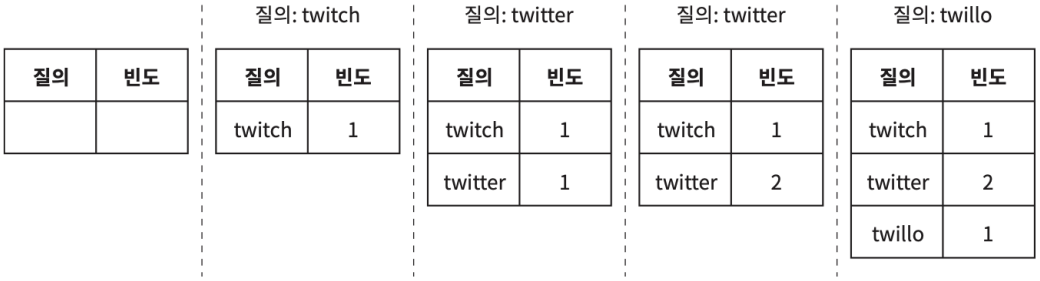
    - 질의문은 평균적으로 4개의 단어로 이루어지고, 각 단어는 평균적으로 다섯 글자로 구성(질의당 평균 20바이트)
- 평균적으로 1회 검색당 20건의 요청이 백엔드로 전달
- 대략 초당 24000건의 질의(QPS)가 발생
    - 천만 사용자 x 10질의 x 20자 / 24시간 / 3600초 = 24000
    - 최대 QPS = 48000
- 질의 가운데 20%는 신규 검색어
    - 대략 0.4GB(천만 사용자 x 10질의 / 일 x 20자 x 20%)
    - 매일 0.4GB의 신규 데이터가 시스템에 추가됨

# ▼ 개략적 설계안 제시 및 동의 구하기

## ▼ 데이터 수집 서비스

사용자가 입력한 질의를 실시간으로 수집하는 시스템

데이터가 많은 애플리케이션에 실시간 시스템을 바람직하지 않음



| 질의 | 빈도 |
|---|---|
| | |

질의: twitch

| 질의 | 빈도 |
|---|---|
| twitch | 1 |

질의: twitter

| 질의 | 빈도 |
|---|---|
| twitch | 1 |
| twitter | 1 |

질의: twitter

| 질의 | 빈도 |
|---|---|
| twitch | 1 |
| twitter | 2 |

질의: twillo

| 질의 | 빈도 |
|---|---|
| twitch | 1 |
| twitter | 2 |
| twillo | 1 |

그림 13-2

빈도 테이블

1. 새로운 질의인 경우 insert 후 빈도수 증가
2. 기존에 저장된 질의인 경우 해당 질의 빈도수 증가

## ▼ 질의 서비스

주어진 질의에 다섯 개의 인기 검색어를 정렬해 내놓는 서비스

| query | freuqency |
|---|---|
| twitter | 35 |
| twitch | 29 |
| twilight | 25 |
| twin peak | 21 |
| twitch prime | 18 |
| twitter search | 14 |
| twillo | 10 |
| twin peak sf | 8 |

**표 13-1**

빈도 테이블

- query
  - 질의문
- frequency
  - 빈도
- 빈도수가 높은 순서로 정렬해서 top 5 조회

```
SELECT * FROM frequency_table
WHERE query Like `prefix%`
ORDER BY frequency DESC
LIMIT 5;
```

  - 데이터 양이 적을 때는 나쁘지 않은 설계

- 데이터가 많아지면 DB 병목 현상 발생

# ▼ 상세 설계

## ▼ 트라이 자료구조

개략적 설계안에서는 관계형 DB를 저장소로 사용했으나 효율적이지 않음

**트라이**를 사용하도록 변경

**트라이**

- 문자열을 꺼내는 연산에 초첨을 맞추어 설계된 자료구조
- 트리 형태의 자료구조
- 루트 노드는 빈 문자열을 나타냄
- 각 노드는 글자 하나를 저장하며 26개의 자식 노드를 가질 수 있음
- 각 트리 노드는 하나의 단어, 또는 접두어 문자열을 나타냄

**트라이 자료구조**

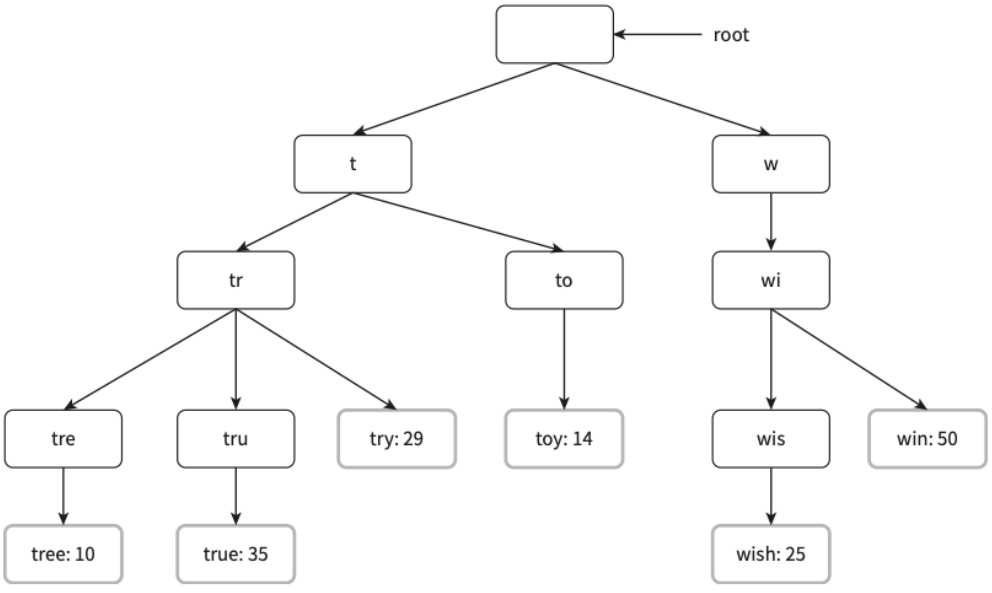



그림 13-6

기본 트라이 자료구조는 노드에 문자들을 저장

정렬된 결과를 내놓기 위해서는 노드에 빈도 정보까지 저장할 필요가 있음

- p → 접두어의 길이
- n → 트라이 안에 있는 노드 개수
- c → 주어진 노드의 자식 개수
- k → 가장 많이 사용된 질의어
    - 해당 접두어를 표현하는 노드를 찾는다 ⇒ 시간복잡도 O(p)
    - 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드(유효한 검색 문자열을 구성하는 노드)를 찾는다 ⇒ 시간복잡도 O(c)
    - 유효 노드들을 정렬하여 가장 인기 있는 검색어 k개를 찾는다 ⇒ 시간복잡도 O(clogc)

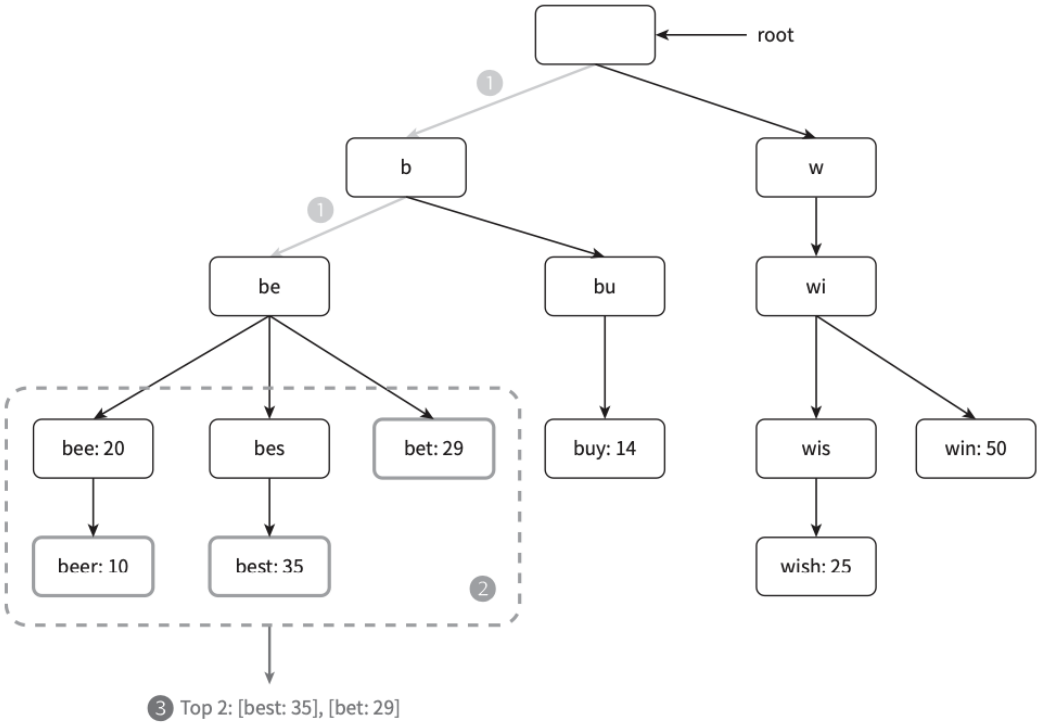



**그림 13-7**

예시

1. 접두어 노드 'be'를 찾음
2. 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드를 찾음
    - beer: 10
    - best: 35

    - bet: 29
3. 유효 노드를 정렬하여 2개만 골라냄
    - best: 35
    - bet: 29

시간 복잡도는 위의 각 단계에 소요된 시간의 합 ⇒ O(p) + O(c) + O(clogc)

최악의 경우에는 k개의 결과를 얻으려고 전체 트라이를 다 검색해야 하는 일이 생길 수 있음

- 접두어 최대 길이 제한
    - 사용자가 검색창에 긴 검색어를 입력하는 일은 거의 없음
    - p값은 작은 정숫값이라고 가정해도 안전
    - 시간복잡도는 O(p)에서 O(작은 상숫값) = O(1)로 바뀔 것
- 노드에 인기 검색어 캐시
    - 각 노드에 k개의 인기 검색어를 저장해 두면 전체 트라이를 검색하는 일을 방지할 수 있음
    - 각 노드에 질의어를 저장할 공간이 많이 필요하게 된다는 단점
    - 빠른 응답속도가 중요할 때는 저장공간을 희상할 만한 가치가 있음

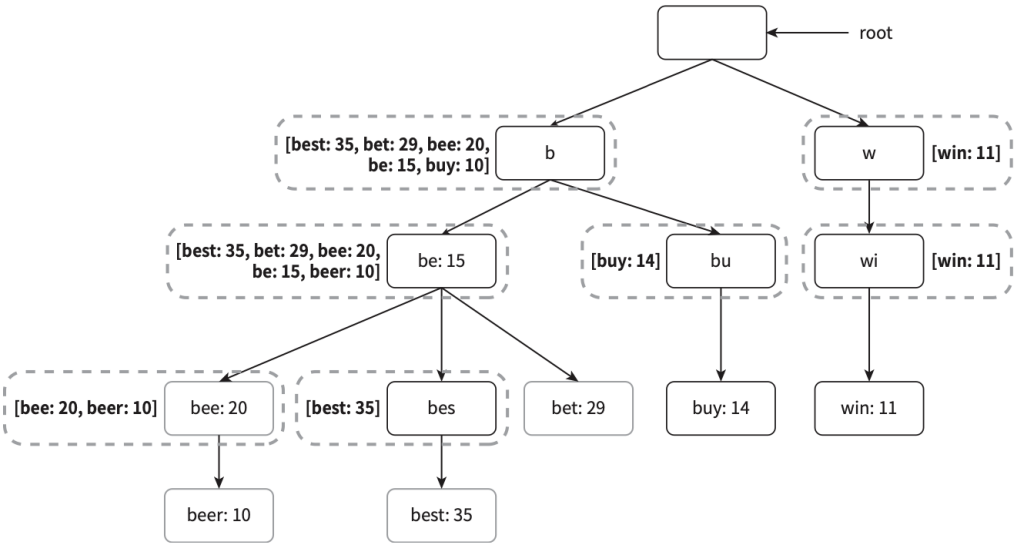



**그림 13-8**

개선된 트라이 구조

- 접두어 노드를 찾는 시간 복잡도 ⇒ O(1)
- 최고 인기 검색어 5개를 찾는 질의의 시간 복잡도 ⇒ O(1)

## ▼ 데이터 수집 서비스

지금까지의 설계는 사용자가 검색창에 뭔가 타이핑을 할 때마다 실시간으로 데이터를 수정함

- 매일 수천만 건의 질의가 입력될 텐데 그때마다 트라이를 갱신하면 질의 서비스는 심각하게 느려질 것
- 일단 트라이가 만들어지고 나면 인기 검색어는 그다지 자주 바뀌지 않을 것 ⇒ 자주 갱신할 필요 없음

실시간 애플리케이션(ex. 트위터)이라면 제안되는 검색어를 항상 신선하게 유지할 필요가 있음

구글 검색 같은 애플리케이션이라면 자주 바꿔줄 필요 없음

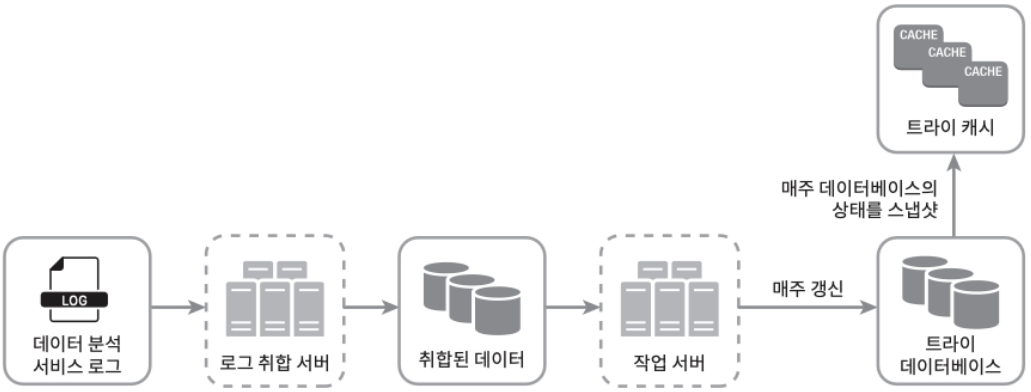



그림 13-9

데이터 분석 서비스의 수정된 설계안

### 데이터 분석 서비스 로그

| query | time |
|---|---|
| tree | 2019-10-01 22:01:01 |
| try | 2019-10-01 22:01:05 |
| tree | 2019-10-01 22:01:30 |
| toy | 2019-10-01 22:02:22 |
| tree | 2019-10-02 22:02:42 |
| try | 2019-10-03 22:03:03 |

표 13-3

검색창에 입력된 질의에 관한 원본 데이터가 보관

새로운 데이터가 추가될 뿐 수정은 이루어지지 않음

### 로그 취합 서버

데이터 분석 서비스로부터 나오는 로그는 양이 엄청나고 데이터 형식도 제각각임

실시간성이 얼마나 중요한지 여부에 따라 취합 주기가 결정됨

실시간 애플리케이션 ⇒ 데이터 취합 주기를 짧게

그 외 대부분의 경우 ⇒ 일주일에 한 번 정도 로그를 취합

### 작업 서버

주기적으로 비동기적 작업을 실행하는 서버 집합

트라이 자료구조를 만들고 트라이 데이터베이스에 저장하는 역할

**트라이 캐시**

분산 캐시 시스템

트라이 데이터를 메모리에 유지하여 읽기 연산 능력을 높임

매주 트라이 데이터베이스의 스냅샷을 떠서 갱신

**트라이 데이터베이스**

지속성 저장소

- 문서 저장소
  - 새 트라이를 매주 만드는 것
  - 주기적으로 트라이를 직렬화하여 DB에 저장 가능
- 키-값 저장소

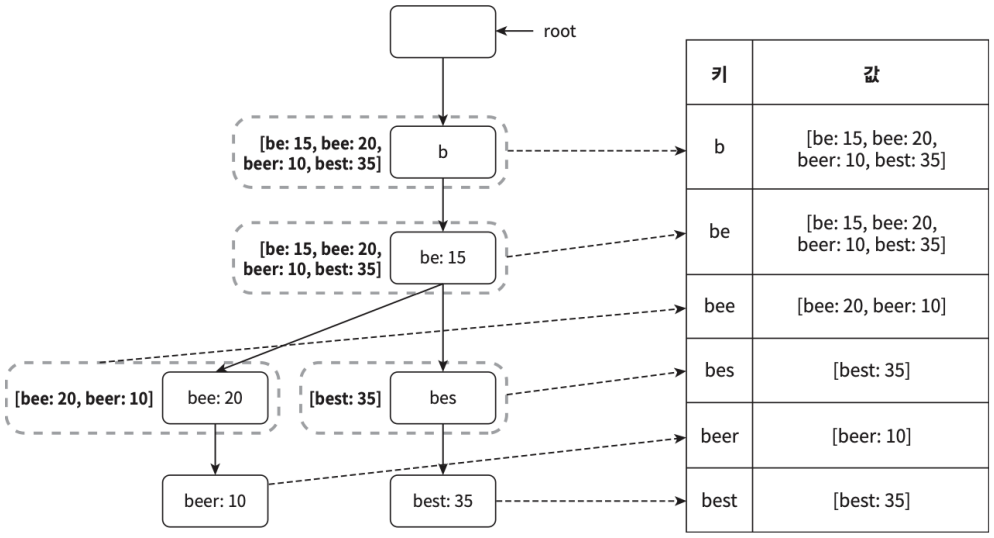



그림 13-10

트라이를 해시 테이블로 대응하는 방법

  - 특정 로직을 사용하여 해시 테이블 형태로 변환 가능
  - 트라이에 보관된 모든 접두어를 해시 테이블 키로 변환
  - 각 트라이 노드에 보관된 모든 데이터를 해시 테이블 값으로 변환

## ▼ 질의 서비스

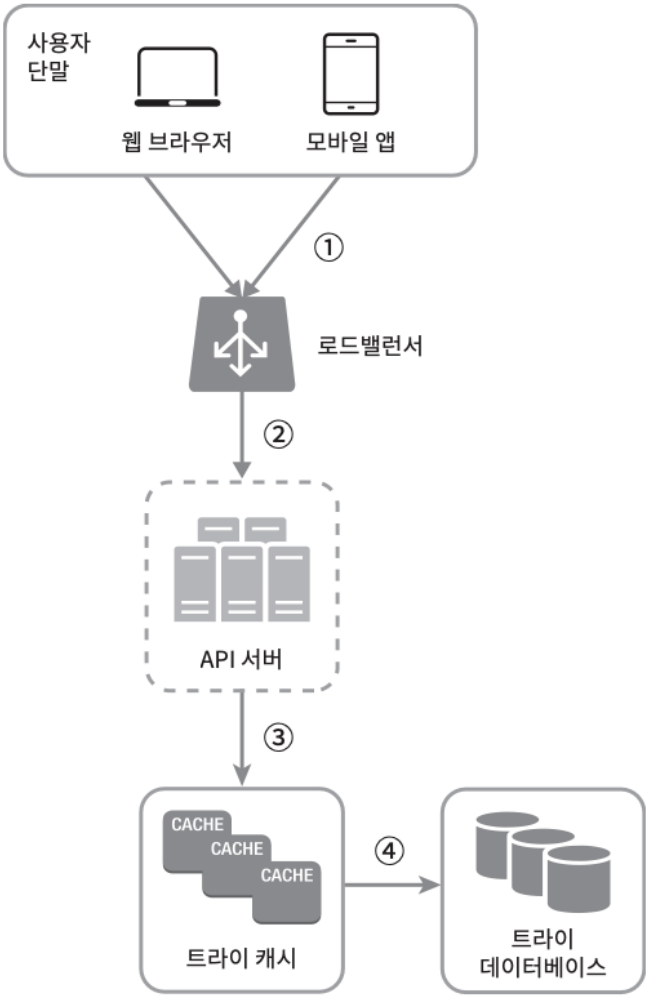



그림 13-11

1. 검색 질의가 로드밸런서로 전송
2. 로드밸런서는 해당 질의를 API 서버로 전송
3. API 서버는 트라이 캐시에서 데이터를 가져와 해당 요청에 대한 자동완성 검색어 제안 응답을 구성
4. 데이터가 트라이 캐시에 없는 경우 데이터를 DB에서 가져와 캐시를 채움
5. 캐시 미스는 캐시 서버의 메모리가 부족하거나 캐시 서버에 장애가 발생할 수 있음

**빠르게 처리하기 위한 최적화 방안**

- AJAX 요청
  - 웹 어플리케이션의 경우 브라우저는 보통 AJAX 요청을 보내서 자동완성된 검색어 목록을 가져옴
  - 요청을 보내고 받기 위해 페이지를 새로고침할 필요가 없다는 장점
- 브라우저 캐싱
  - 대부분 애플리케이션의 경우 자동완성 검색어 제안 결과는 짧은 시간 안에 자주 바뀌지 않음
  - 제안된 검색어들을 브라우저 캐시에 넣어두면 후속 질의의 결과는 해당 캐시에서 바로 가져갈 수 있음
  - 구글 검색 엔진이 이런 캐시 메커니즘을 사용함
- 데이터 샘플링
  - 대규모 시스템의 경우 모든 질의 결과를 로깅하도록 해 놓으면 CPU 자원과 저장공간을 엄청나게 소진하게 됨
  - 데이터 샘플링 기법 ⇒ N개 요청 가운데 1개만 로깅하도록

## ▼ 트라이 연산

**트라이** → 검색어 자동 완성 시스템의 핵심 컴포넌트

### 트라이 생성

- 작업 서버가 담당
- 데이터 분석 서비스의 로그나 DB로부터 취합된 데이터 이용

### 트라이 갱신

1. 매주 한 번 갱신하는 방법
   - 새로운 트라이를 만든 다음에 기존 트라이를 대체
2. 트라이의 각 노드를 개별적으로 갱신하는 방법
   - 성능이 좋지 않지만 트라이가 작을 때는 고려해봄직한 방안
   - 상위 노드에도 인기 검색어 질의 결과가 보관되기 때문에 트라이 노드를 갱신할 때는 그 모든 상위 노드를 갱신해야 함

**검색어 삭제**

- 여러 가지로 위험한(혐오성이 짙거나, 폭력적이거나, 성적으로 노골적이거나) 질의어는 자동완성 결과에서 제거해야 함
- 트라이 캐시 앞에 필터 계층을 두고 부적절한 질의어가 반환되지 않도록 함

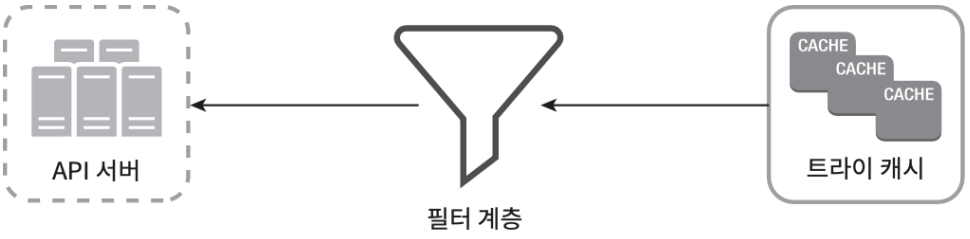



그림 13-14

## ▼ 저장소 규모 확장

트라이의 크기가 한 서버에 넣기엔 너무 큰 경우에도 대응할 수 있도록

첫 글자를 기준(영어만 지원하면 되기 때문에)으로 샤딩하는 방법

- 서버 2개
  - 'a' ~ 'm' 은 첫 번째 서버에 저장
  - 'n' ~ 'z' 는 두 번째 서버에 저장
- 서버 3개
  - 'a' ~ 'i' 은 첫 번째 서버에 저장
  - 'j' ~ 'r' 는 두 번째 서버에 저장
  - 's' ~ 'z' 는 세 번째 서버에 저장
- 서버 4개 ~
  - 계층적으로 샤딩해야 함
  - ex) 'aa' ~ 'ag' / 'ah' ~ 'an' / 'ao' ~ 'au' ~ ...
    - 그럴싸해 보이지만 'c'로 시작하는 단어가 'x'로 시작하는 단어보다 많아서 데이터를 서버에 균등하게 분배하기 불가능

  **⇒ 과거 질의 데이터의 패턴을 분석하여 샤딩**

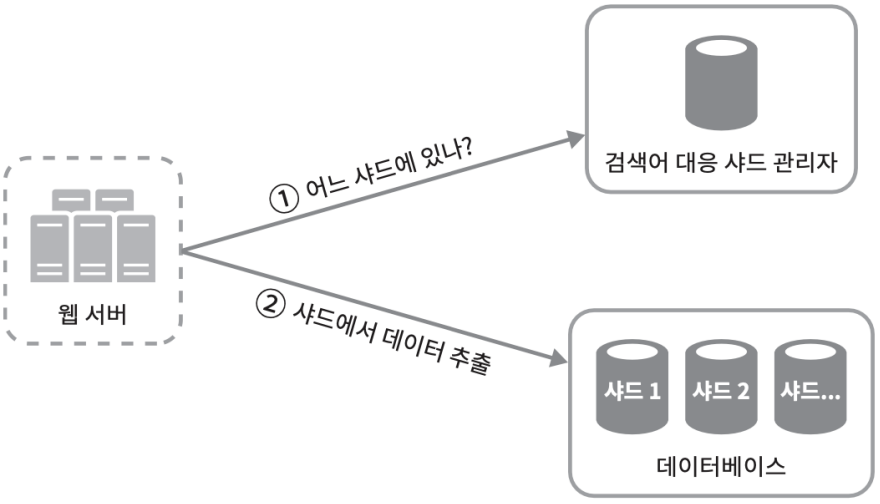



그림 13-15

## ▼ 마무리

**다국어 지원이 가능하도록 시스템을 확장하려면**

트라이에 유니코드 데이터를 저장

**국가별로 인기 검색어 순위가 다르다면**

국가별로 다른 트라이를 사용

트라이를 CDN에 저장하여 응답속도 높이기

**실시간으로 변하는 검색어의 추이를 반영하려면**

현 설계안은 적합하지 않음

- 작업 서버가 매주 한 번씩만 돌도록 되어 있음
- 때맞춰 실행된다고 해도 너무 많은 시간이 소요됨

책에서 다룰 수 있는 범위를 넘어섬

- 샤딩을 통해 작업 대상 데이터의 양을 줄임
- 순위 모델을 바꾸어 최근 검색어에 보다 높은 가중치를 줌
- 한번에 모든 데이터를 동시에 사용할 수 없을 가능성이 있다는 점을 고려
  - 데이터 스트리밍이 됨 = 데이터가 지속적으로 생성됨
  - 스트림 프로세싱에는 특별한 시스템이 필요

## ▼ 토론

검색어 필터링을  어떤 식으로 하면 좋을지? 필터링 조건들을 하나씩 수동으로 지정해야 하는지? 사용자의 피트백을 받으면 좋을 것 같은데 이건 어떻게 적용할 수 있을지? 고민