7장 - 분산 시스템을 위한 유일 ID 생성기 설계

- 이번 장은 분산 시스템에 사용될 유일 ID 생성기를 설계

auto_increment 속성이 설정된 RDB의 기본 키를 사용할 수 없음 (분산 환경이니까)
RDB 안대로는 해당 요구를 감당할 수 없을 뿐더러, 지연 시간을 낮추기가 매우 힘듦

1단계 : 문제 이해 및 설계 범위 확정

```
지원자 : ID는 어떤 특성을 갖나요?

면접관 : ID는 유일해야 하고, 정렬 가능해야 합니다.

지원자 : 새로운 레코드에 붙일 ID는 항상 1만큼 큰 값이어야 하나요?

면접관 : ID의 값은 시간이 흐름에 따라 커질 테지만 언제나 1씩 증가한다고 할
       다만 확실한 것은, 아침에 만든 ID보다는 저녁에 만든 ID가 큰 값을 갖

지원자 : ID는 숫자로만 구성되나요?

면접관 : 그렇습니다.

지원자 : 시스템 규모는 어느 정도입니까?

면접관 : 초당 10,000 ID를 생성할 수 있어야 합니다.
```

> 요구사항을 정리하자면 다음과 같다.

- ID는 유일해야 한다.
- ID는 숫자로만 구성되어야 한다.
- ID는 64비트로 표현될 수 있는 값이어야 한다.
- ID는 발급 날짜에 따라 정렬 가능해야 한다.

- 초당 10,000개의 ID를 만들 수 있어야 한다.

2단계 : 개략적 설계안 제시 및 동의 구하기

분산 시스템에서 유일성이 보장되는 ID를 만드는 선택지

- **다중 마스터 복제(multi-master replication)**
- **UUID(Universally Unique Identifier)**
- **티켓 서버(ticket server)**
- **트위터 스노플레이크(twitter snowflake) 접근법**

다중 마스터 복제(multi-master replication)

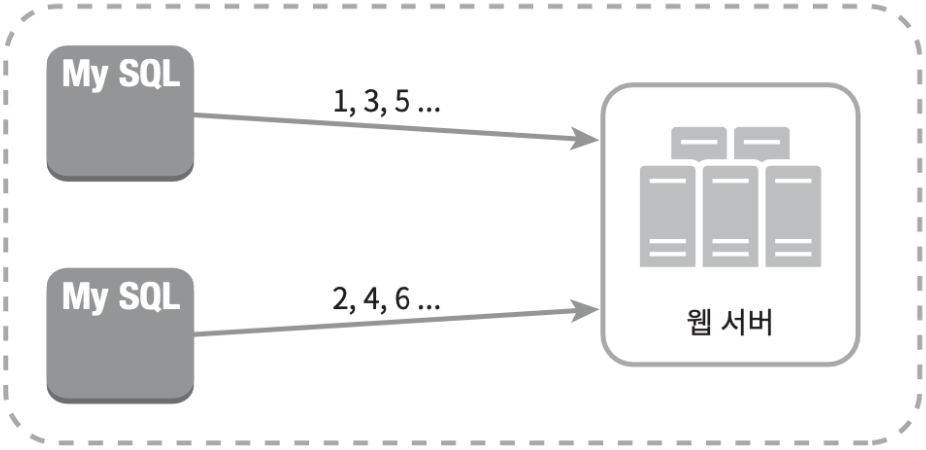

그림 7-2

- 데이터베이스의 `auto_increment` **기능을 활용**하는 방식
  다음 ID의 값을 구할 때
  1만큼 증가시키는 것이 아니라 `k` **만큼 증가** (`k` = 사용중인 데이터베이스 서버 수)
- 이렇게 하면 규모 확장성 문제를 어느 정도 해결할 수 있다.
  - 데이터베이스 수를 늘리면 초당 생성 가능 ID 수도 늘릴 수 있음
- **큰 단점**
  - 여러 데이터 센터에 걸쳐 규모를 늘리기 어렵다.
  - ID의 유일성은 보장되지만 그 값이 시간 흐름에 맞추어 커지도록 보장할 수 없음

- 서버를 추가하거나 삭제할 때도 잘 동작하도록 만들기 어려움

UUID(Universally Unique Identifier)

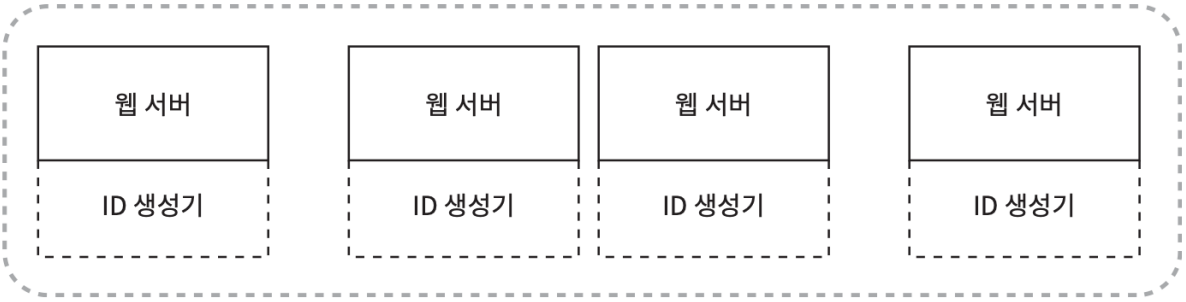

그림 7-3

- 컴퓨터 시스템에 저장되는 정보를 유일하게 식별하기 위한 128비트짜리 수
- 충돌 가능성이 지극히 낮음
  - 중복 UUID가 1개 생길 확률을 50%로 올리려면 **초당 10억 개의 UUID를 100년 동안 계속** 만들어야 됨
- 서버간 조율 없이 독립적으로 생성 가능 (각자의 웹 서버에서 UUID를 만들어내면 됨)
- **장점**
  - UUID를 만드는 것이 단순, 서버 사이의 조율이 필요 없음으로 동기화 이슈도 없음
  - 각 서버가 자기가 쓸 ID를 알아서 만드는 구조이므로 규모 확장도 쉬움
  - 몽고의 기본 ID 생성이 UUID인 것으로 알고 있음
- **단점**
  - ID가 128비트로 길다. (이번 장에서 요구하는 요구사항은 64비트)
  - ID를 시간 순으로 정렬할 수 없음
  - ID에 숫자가 아닌 값이 포함될 수 있음

티켓 서버(ticket server)

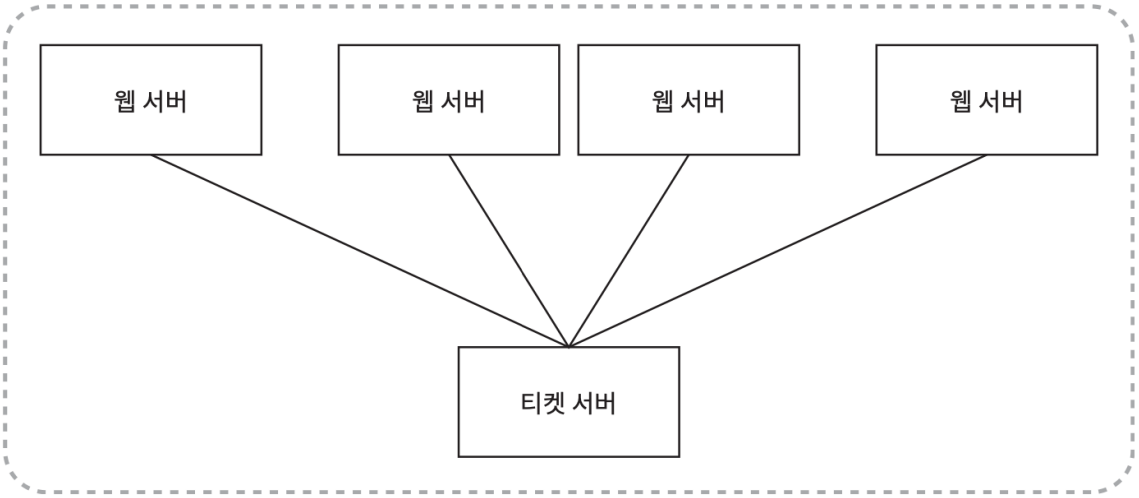

그림 7-4

- 플리커(Flickr)는 분산 기본 키를 만들어내기 위해 이 기술을 사용
- 이 아이디어의 핵심은 auto_increment 기능을 갖춘 데이터베이스 서버,
  즉
  티켓 서버를 중앙 집중형으로 하나만 사용하는 방식
- **장점**
  - 유일성이 보장되는 오직 숫자로만 구성된 ID를 쉽게 만들 수 있음
  - 구현하기 쉽고, 중소 규모 애플리케이션에 적합
- **단점**
  - **티켓 서버가 SPOF**가 됨
    이 서버에 장애가 발생하면, 해당 서버를 이용하는 모든 시스템이 영향을 받음
    이 이슈를 피하려면 티켓 서버를 여러 대 준비해야 됨 (이렇게 할 경우 동기화 이슈가 발생할 수 있음)

트위터 스노플레이크(twitter snowflake) 접근법

- 이 방식은 독창적인 ID 생성 기법을 사용함
  - **분할 정복(divide and conquer)**을 사용

그림 7-5

- 사인(sign) 비트 : 1비트 할당, 지금 쓰임새는 없지만 나중을 위해 유보해 둠(음수 양수 구분)
- 타임스탬프(timestamp): 41비트 할당. 기원 시각(epoch) 이후로 몇 및밀리초가 경과했는지 나타냄
- 데이터센터 ID : 5비트 할당. - 2 ** 5 = 32개 데이터센터를 지원할 수 있음
- 서버 ID: 5비트 할당. - 데이터 센터당 32개 서버를 사용할 수 있음
- 일련번호 : 12비트 할당 - 각 서버에서 ID를 생성할 때마다 일련번호를 1만큼 증가
    - 1밀리초가 경과할 때마다 0으로 초기화

3단계 : 상세 설계

타임스탬프

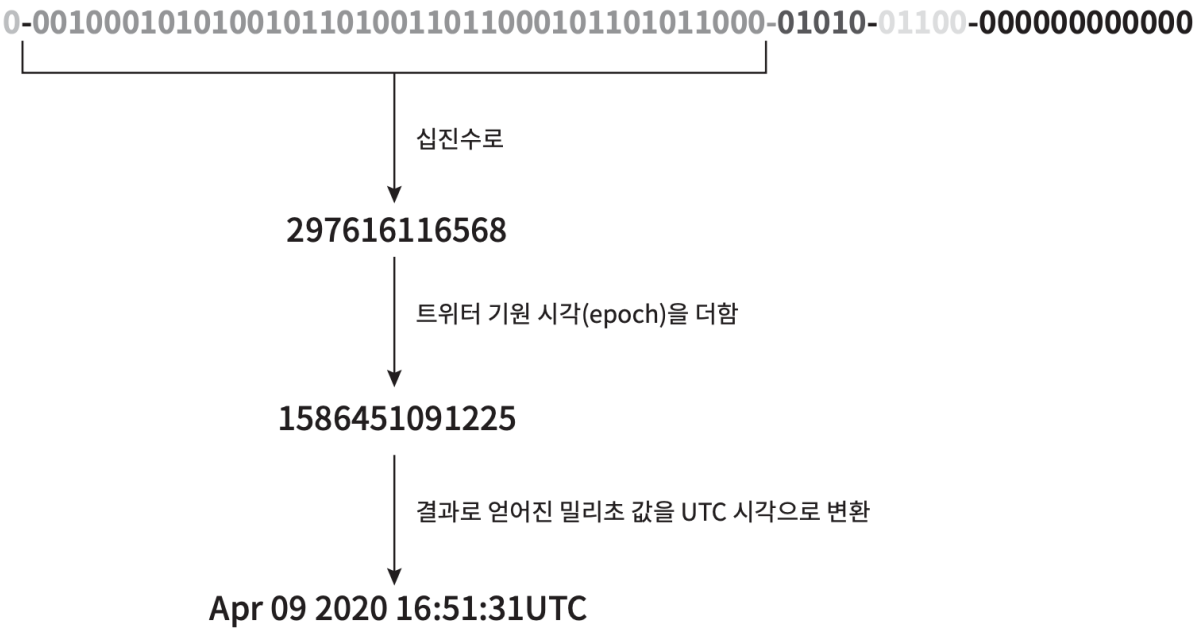

그림 7-7

- ID 구조에서 가장 중요한 41비트 차지

- 시간이 흐름에 따라 점점 큰 값을 갖게 됨 (ID는 시간 순으로 정렬 가능)
- 41비트로 표현할 수 있는 타임스탬프의 최대값은 2^41-1 = 2199023255551밀리초
  - 이 값은 대략 69년에 해당 (2199023255551밀리초 / 1000 / 365일 / 24시간 / 3600초)
  - 따라서 이 ID 생성기는 69년동안만 정상 동작
  - 기원 시각을 현재에 가깝게 맞춰 오버플로(overflow)가 발생하는 시점을 늦춰 놓은 것
  - 69년이 지나면 기원 시각을 바꾸거나 ID 체계를 다른 것으로 이전(migration)해야 함

일련 번호

- 12비트 이므로 2^12 = 4096개의 값을 가질 수 있음
- 어떤 서버가 같은 밀리초 동안 하나 이상의 ID를 만들어 낸 경우에만 0보다 큰 값을 갖게 됨

4단계 : 마무리

- 다중 마스터 복제, UUID, 티켓 서버, 트위터 스노플레이크의 네 방법을 살펴봄
- **이 장에서 선택한 방식**은 `스노플레이크`, 모든 요구사항을 만족하면서 분산 환경에서 규모 확장 가능

면접관과 추가로 논의할 만한 내용

- **시계 동기화(clock synchronization)**
  - ID 생성 서버들이 전부 같은 시계를 사용한다고 가정했지만, 하나의 서버가 여러 코어에서 실행될 경우 유효하지 않음
  - 여러 서버가 물리적으로 독립된 여러 장비에서 실행되는 경우에도 마찬가지
  - **NTP(Network Time Protocol)**은 이 문제를 해결하는 가장 보편적인 수단 (여기 링크 참고)
- **각 절(section)의 길이 최적화**

    - **동시성(concurrency)이 낮고 수명이 긴 애플리케이션**이라면
      일련번호 절의 길이를 줄이고 타임스탬프 절의 길이를 늘리는 것이 효과적

- **고가용성(high availability)**
    - ID 생성기는 필수 불가결(mission critical) 컴포넌트이므로 아주 높은 가용성을 제공해야 함

▼ Chapter 7: DESIGN A UNIQUE ID GENERATOR IN DISTRIBUTED SYSTEMS

1. Universally unique identifier
2. Ticket Servers: Distributed Unique Primary Keys on the Cheap
3. Announcing Snowflake
4. Network Time Protocol