주요 키 채번 전략

Overview: Choosing an appropriate database primary key

2025.09.24.

System Software Lab.

목차

Primary key

프라이머리 키, 주 키, 주요 키, 기본 키

후보 키

데이터베이스 관리자가 선택한 데이터베이스의 테이블 내 레코드를 유일하게 구분할 수 있는 식별자 중 최소한의 속성을 가진 식별자

슈퍼 키

- 실무에서는 학번, 주민등록번호, 휴대폰번호 등 자연 키*(Natural key)*를 사용하지 않고 별도의 키를 생성하는 전략을 많이 사용
  - 이러한 별도의 키를 인조 키 또는 대체 키*(Surrogate key)* 라고 함
  - 자연 키는 대체로 그 값이 변경될 가능성이 큼
  - 자연 키는 대체로 단일한 값만으로 레코드를 구분할 수 없어 여러 값을 묶어야 하는데*(이를 복합 키라고 함)*

    이런 경우 개발 난도가 상승하고 성능 저하가 발생할 수 있음

Auto increment *(자동 증가)*

< 주요 키 >	이름	결제 일시	결제 품목	휴대폰번호
10	김준옥	2025-03-12T23:30:26	A12N	013-1001-0011
11	박성수	2025-04-01T11:30:03	C763D	013-0101-1000
12	김지혜	2025-04-01T11:30:05	J432	013-0010-0100
13	김미지	2025-06-30T06:25:32	K0E1D	013-0001-0001
14	고채은	2025-07-07T19:25:38	X4DG	013-1000-1111
20	이영수	2025-09-24T02:12:54	J378SD6	013-1100-1010

- 데이터베이스가 1부터 레코드가 삽입될 때마다 자동으로 증가하여 채번하여 주요 키를 생성
- 분산 데이터베이스 환경에서 주요 키가 중복될 수 있음
- 사용자가 데이터 개수를 예측하거나, IDOR*(안전하지 않은 직접 객체 참조)*이 가능할 수 있음
- 데이터베이스 자원을 소모해 주요 키를 자동 삽입함
- 대용량 데이터 삽입 시 매번 데이터베이스로부터 다음 주요 키 값을 받아와야 함

Auto increment *(자동 증가)*

분산 데이터베이스 환경에서 주요 키가 중복될 수 있다

우아한배달요 데이터베이스

서울특별시	경상북도	대구광역시	강원특별자치도
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6

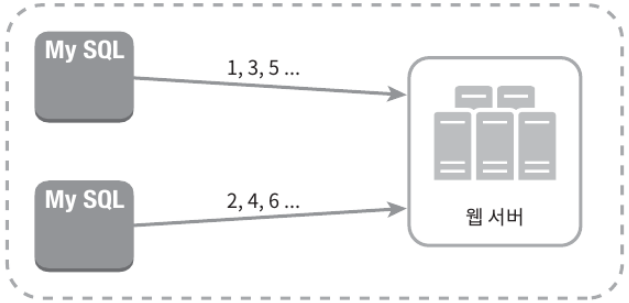
- 지역별로 데이터베이스를 나누어 결제 내역을 저장하는 서비스가 있다고 가정
- "지역별로 구분하는 것은 불편하다"며 서비스 요구사항이 변경되며 데이터베이스를 통합해야 할 필요가 발생하면?
- 특정 지역의 결제가 많아 특정 지역의 데이터베이스를 추가로 늘려야 할 필요가 발생하면?
- 각 레코드의 주요 키가 중복돼 데이터베이스 유일성을 위반할 수 있음

Auto increment *(자동 증가)*

분산 데이터베이스 환경에서 주요 키가 중복될 수 있다

AUTO_INCREMENT를 사용하며 이를 해결하는 방법도 있음

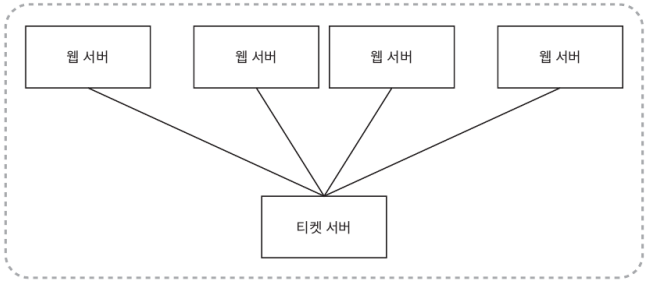
다중 마스터 복제



- 다음 ID를 구할 때 1만큼 증가하는 것이 아닌 데이터베이스 개수만큼 증가시킴
- 데이터베이스 개수를 늘리거나 줄이기 어렵다.
- ID 값이 시간의 흐름에 따라 커짐을 보장하기 어렵다.

  레코드 A, B, C가 각각 서버 1, 1, 2에 저장되는 경우 A는 1번, B는 3번, C는 2번이 되어 버린다.

티켓 서버



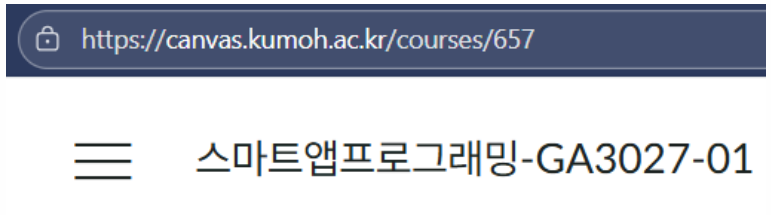
- 다음 ID를 구할 때 ID가 1씩 증가하는 티켓 서버에서 ID를 받아옴
- 티켓 서버에 장애가 발생*(단일 장애점)*하면 모든 시스템이 마비될 수 있음

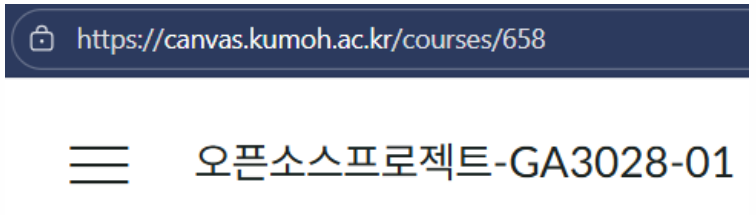
  티켓 서버를 여러 대 준비하여 해결할 수 있으나 데이터 동기화 등 새로운 문제가 발생한다.

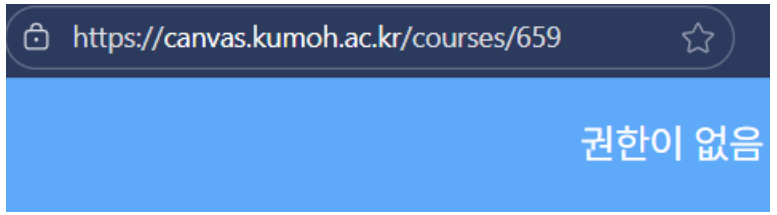
- 여러 서버에서 티켓 서버로 요청 시 병목 현상이 발생할 수 있음

Auto increment *(자동 증가)*

사용자가 데이터 개수를 예측하거나, 안전하지 않은 객체 직접 참조가 가능할 수 있음







- 공격자가 URL 등의 경로에서 주요 키 패턴을 쉽게 예측하여 다른 사용자의 데이터에 접근할 수 있음

<이화여자대학교>

'24. 9. 2.(월) ~ 9. 3.(화) 해커가 시스템의 데이터베이스(DB) 조회 기능의 취약점*을악용한파라미터 변조 공격으로 이화여자대학교 통합행정시스템에 침입하여 **8만 3천여 명**의 주민등록번호를 포함한 **개인정보**를 **탈취**하였다.

* 개인정보 조회 시, 세션값(사용자 식별값)과 조회 대상 정보가 불일치하는 경우에도 파라미터(학번) 변조를 통해 다른 사용자의 개인정보 조회가 가능한 취약점 존재

개인정보보호위원회 보도자료 ('25.06.12.)
안전조치 소홀로 개인정보 유출한 2개 대학에 과징금 9억 6,600만 원 부과

UUID *(Universally Unique Identifier)* / GUID *(Globally Unique Identifier)*

- 128비트로 구성된 BASE16으로 인코딩된 ID로, 버전에 따라 아이디 생성 방식이 다름
- ID 생성을 위해 각 서버가 통신할 필요가 없음
- 8a463aa4-b1dc-4f27-9c3f-53b94dc45e74처럼 8-4-4-4-12 패턴으로 구성됨
- 일반적으로 UUIDv4가 주로 사용되며 이 버전은 중복 UUID가 생길 확률을 50%가 되게 하려면 2.71×10^{18} 개(퀸틸리온)를 만들어야 함
  - 초당 10억 개의 UUID를 100년 동안 계속해서 만들어야 하는 수준
- Java가 기본으로 채택한 UUIDv4는 무작위로 생성돼 시간 순으로 정렬할 수 없으며 알파벳도 포함됨

< 주요 키 >
3f1e2d0c-7b92-1abc-9d44-12f3a4b5c6d7
4c2a10ff-98ad-2b30-9a6b-5577cc88ee11
f47ac10b-58cc-4372-a567-0e02b2c3d479
01890f2a-7c9e-7b0d-9c11-001122334455
5e2f3a9c-1d4b-5c0e-8f77-112233445566

UUID *(Universally Unique Identifier)*

- 128비트로 구성된 BASE16으로 인코딩된 ID로, 버전에 따라 아이디 생성 방식이 다름
- ID 생성을 위해 각 서버가 통신할 필요가 없음

UUIDv1

- 타임스탬프*, 네트워크 인터페이스 MAC 주소, 무작위 번호를 사용해 ID를 생성
- UUID가 생성된 시기와 위치를 추적할 수 있음
- MAC 주소와 시각이 변조될 수 있어 전 세계적으로 고유함이 보장되지 않음

UUIDv2

- UUIDv1의 현재 시간, MAC 주소, 무작위 번호와 함께 UUID를 생성하는 프로세스의 번호도 사용해 ID를 생성
- 여전히 전 세계적으로 교유함이 보장되지 않지만 UUIDv1에 비해 그 가능성이 더 낮음

* 그레고리력 도입일(1582년 10월 15일 00시 00분 00초)을 기준으로 100ns씩 증가하는 타임스탬프

UUID *(Universally Unique Identifier)*

- 128비트로 구성된 BASE16으로 인코딩된 ID로, 버전에 따라 아이디 생성 방식이 다름
- ID 생성을 위해 각 서버가 통신할 필요가 없음

UUIDv3

- 미리 정의된 네임스페이스 중 하나를 선택하고, 문자열을 입력하고 이를 MD5 해시를 사용해 ID를 생성
- 같은 네임스페이스와 같은 문자열을 사용하면 항상 동일한 ID가 생성됨

UUIDv4

- 완전 무작위로 ID를 생성
- 언제, 어디서 생성되었는지 추적할 수 없음

UUIDv5

- UUIDv3와 유사하지만 SHA-1 해시를 사용해 ID를 생성
- 128비트를 출력하는 MD5와 달리 160비트 길이와 복잡한 압축으로 SHA-1의 경우 동일한 ID를 만들기 어려움

UUID *(Universally Unique Identifier)*

- 128비트로 구성된 BASE16으로 인코딩된 ID로, 버전에 따라 아이디 생성 방식이 다름
- ID 생성을 위해 각 서버가 통신할 필요가 없음

UUIDv6

- 타임스탬프*와 무작위 번호를 사용해 ID를 생성
- UUIDv1과 유사하지만 MAC 주소를 포함하지 않아 전체적으로 충돌 가능성을 낮춤

UUIDv7

- 유닉스 타임스탬프**와 무작위 번호를 사용해 ID를 생성

UUIDv8

- UUIDv4와 비슷하지만 개발자가 시스템 구조에 맞게 추가 정의하여 ID를 생성

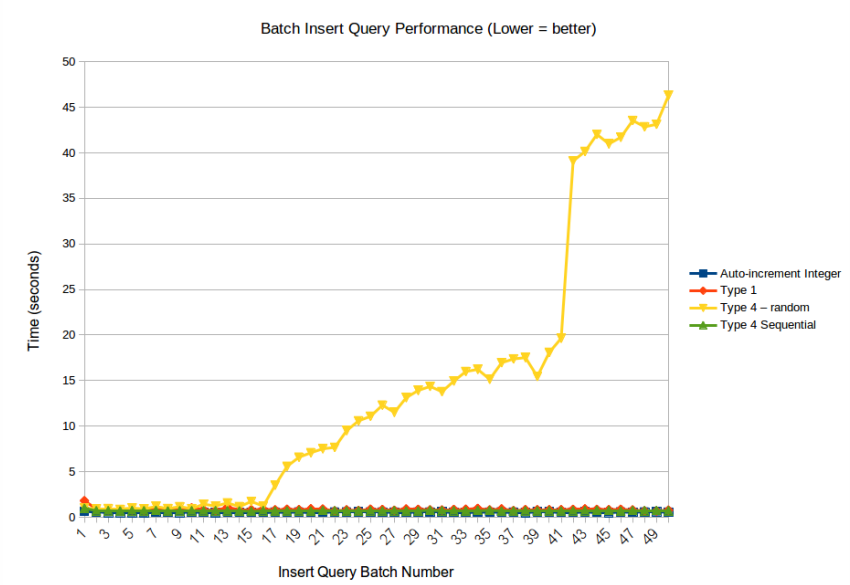
* 그레고리력 도입일(1582년 10월 15일 00시 00분 00초)을 기준으로 100ns씩 증가하는 타임스탬프
** 1970년 1월 1일 00시 00분 00초를 기준으로 1ms씩 증가하는 타임스탬프

UUID *(Universally Unique Identifier)*

많이 사용하는 UUIDv4와 UUIDv7에 대해 더 알아보기

UUIDv4

- 완전 무작위로 ID를 생성
- 새로운 데이터가 인덱스에 순차적으로 삽입되지 않고 무작위 위치에 삽입되므로 I/O 비용이 증가하고 데이터 조회 속도가 느려질 수 있음



https://blog.programster.org/mysql-performance-when-using-uuid-for-primary-key

UUIDv7

- 유닉스 타임스탬프와 무작위 번호를 사용해 ID를 생성
- 시간 순서로 정렬할 수 있으며 새로운 데이터가 인덱스에 순차적으로 삽입돼 인덱스 효율성이 높음
- 타임스탬프가 ID에 포함돼 일반 사용자에게 노출 시 의도치 않은 생성 시각 노출이 발생할 수 있음
  - 예를 들면 환자의 방문일, 약 처방일 등이 노출될 수 있음
  - 유럽연합 일반 데이터 보호 규칙*(GDPR)* 위반 사항이 될 수 있음

ULID *(Universally Unique Lexicographically Sortable Identifier)*

- 비트로 구성된 ID로, UUID의 무작위 생성으로 인한 사전순 정렬 단점을 개선하기 위해 만들어짐
- 유닉스 타임스탬프와 무작위 번호를 사용해 ID를 생성
- UUID는 BASE16을 사용하지만 ULID는 BASE64를 사용하여 각 문자가 4비트가 아닌 5비트를 차지함
- 효율성과 가독성을 위해 UUID와 달리 특수문자 및 I, L, O, U가 제외됨

< 주요 키 >
01K5V5D961BQS8TXCNBQKAEYH6
01K5V5D9613CFXZ7XKH31M3G15
01K5V5D961VMY56RVKG7429761
01K5V5D96167P7SDAWTX80JZHR
01K5V5D961VKS9KE947X1QN0PA

❄ Snowflake ID

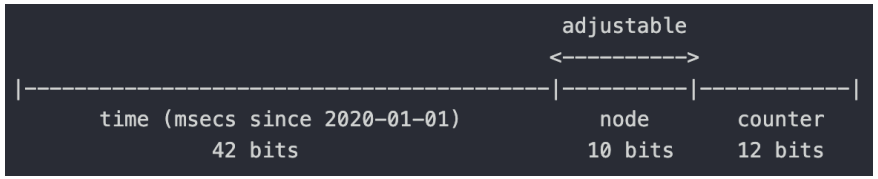
- 2010년 Twitter에서 분산 시스템에서 유일한 ID를 생성하기 위한 방법으로 제안한 10진 64비트 ID
  - UUID, ULID가 128비트, 문자열을 사용하는 것과 달리 64비트 정수형으로 이루어짐
- 더 오랫동안 사용하기 위해*(오버플로우가 도래하는 시간을 지연시키기 위해)* 채번 시작 시간부터의 유닉스 타임스탬프를 사용
  - X*(구 Twitter)*는 채번 시작 시간이 1288834974657(2010년 11월 4일 10시 42분 54초)라고 함
  - 2조 2천 억ms까지 표현 가능하므로 약 70년까지 Snowflake ID를 사용할 수 있음
  - 위UUIDv7, ULID는 1970년부터 8919년 지난 10889년까지 사용할 수 있음

< 주요 키 >
1709823412678899200
1715532098142769152
1820043317659084801
1897742203345190912
1993321187990458368

TSID *(Time-Sorted Unique Identifier)*

- ULID와 Snowflake ID의 장점을 합친 브라질 개발자 Fabio Lima가 만든 BASE32 64비트 ID
- Snowflake ID와 구조가 동일하나 ULID의 무작위성이 추가됨
  - 동일한 서버라도 매 요청마다 node 값이 달라지나, 설정을 통해 유지할 수도 있음
  - counter는 Snowflake ID는 순차 증가였으나 TSID는 무조건 무작위로 결정됨

< 주요 키 >
4472945381276409 1
4472945381281963 7
4472945381907720 1
4472945390021189 5
4472945397773061 2



정리

- Auto increment, UUID, ULID, Snowflake ID, TSID에 대해 알아보았으며 Flake ID, Nano ID, shortid, TESID, Sqids 등 다양한 ID 생성 규칙이 존재
- 규모가 작거나 확장 가능성이 작거나, 정보 노출에 민감하지 않은 서비스에서는 Auto increment를 사용
  - 데이터베이스에서 키를 생성해 간단하게 구현할 수 있음
- 규모가 크거나 확장 가능성이 있거나, 정보 노출에 민감한 서비스에서는 Snowflake ID 또는 TSID 등을 사용
  - 문자열이 포함된 UUID, ULID와 달리 64비트 정수형을 사용하여 시간 기반으로 정렬할 수 있음
  - 다만 타임스탬프를 기반으로 이루어져 사용자가 언제 레코드가 생성되어 있는지 알 수 있어 보안 상 위험할 수 있음
    - 일각에서는 호들갑, 괜한 걱정이라고 함
- 최근에 UUIDv47이라고 해서 홍콩의 개발자가 데이터베이스 내부에는 UUIDv7을, 외부에는 UUIDv4처럼 변환하는 MIT 라이선스의 오픈소스를 출시하기도 함
- 사실 대규모의 사용자가 동시에 이용하는 서비스 외에는 Auto increment와 확실한 권한 관리를 사용하는 것이 적절하다고 판단
  - 퀵 정렬 고안자 영국의 컴퓨터 과학자 Tony Hoare의 격언 "조기 최적화는 모든 악의 근원이다"처럼 스케일 아웃을 상정하고 데이터베이스를 설계하기 보다 **필요하면 주요 키를 바꾸는 전략**을 사용

들어 주셔서 감사합니다

E-mail	park@duck.com
LinkedIn	yeonjong-park
Instagram	yeonjong.park
GitHub	patulus

2025.09.24.

System Software Lab.