DB 분산락도 락이다

7기 BE 투다

목차

개요

우아한기술블로그 개발자 채용 ↗ 뉴스레터 신청

WMS 재고 이관을 위한 분산 락 사용기

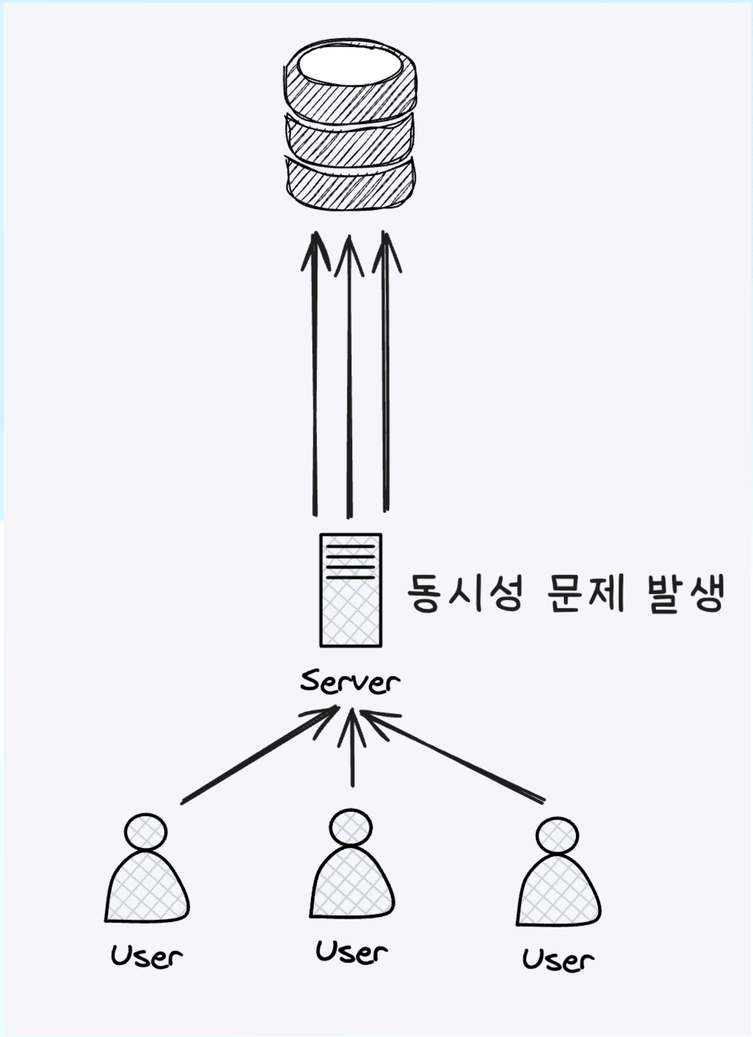
2024. 5. 28. 김준홍 

Backend

> WMS 재고 이관 과정에서 발생한 동시성 이슈를 분산 락(Distributed Lock)을 사용해 해결한 경험을 공유하는 글입니다. 본 글은 분산 락에 대해 알고 있는 분들을 대상으로 작성되었습니다. 제가 경험한 내용들이 여러분들의 비즈니스에 도움이 되는 글이 되길 바랍니다.

락은 아는데 분산락은 뭐지?

락



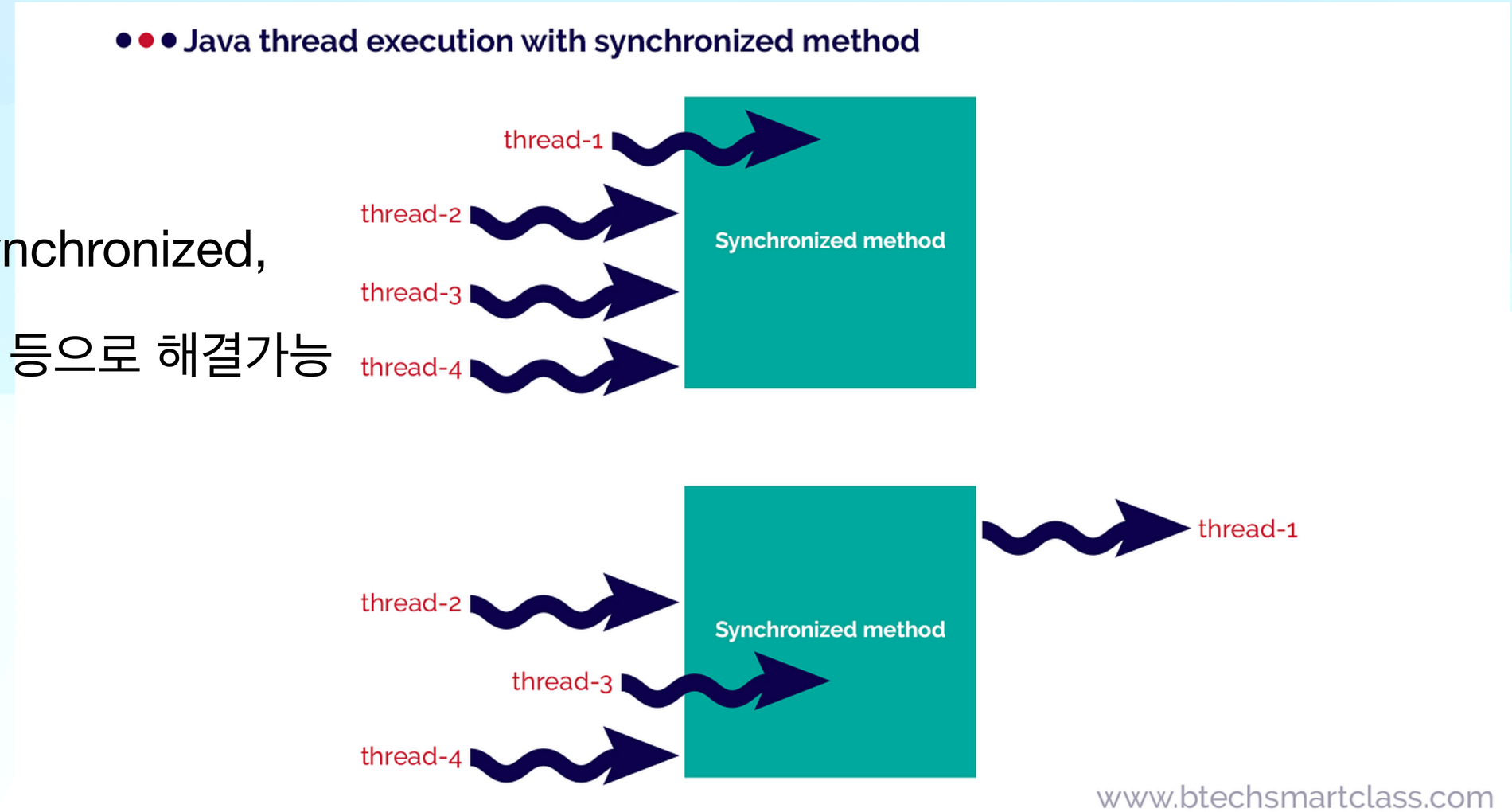
ex) 계좌에 만원 있는데 만원짜리 요청이 3번통과

-3만원이됨

락

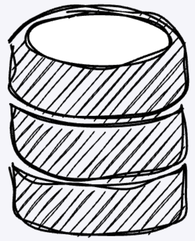
단일 환경에서는 synchronized,

동시성 컬렉션 사용 등으로 해결가능



락

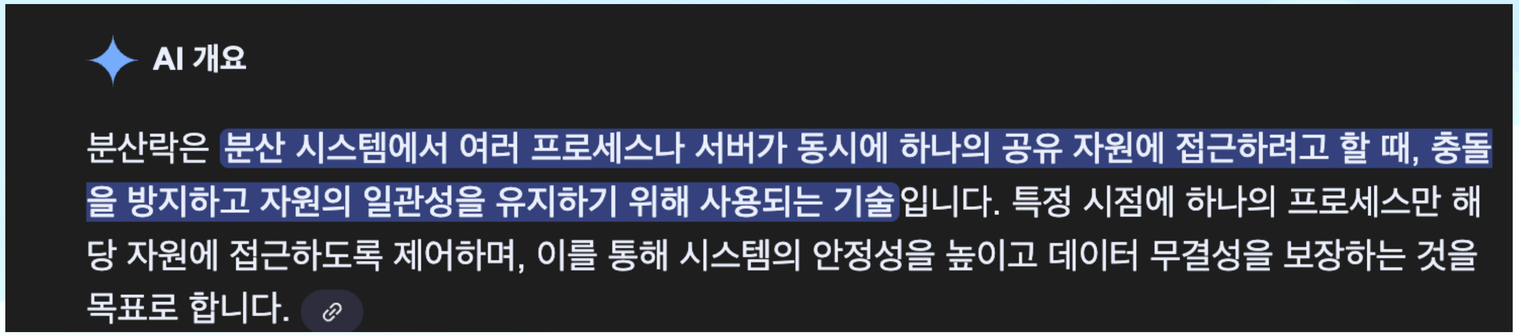
공유 자원



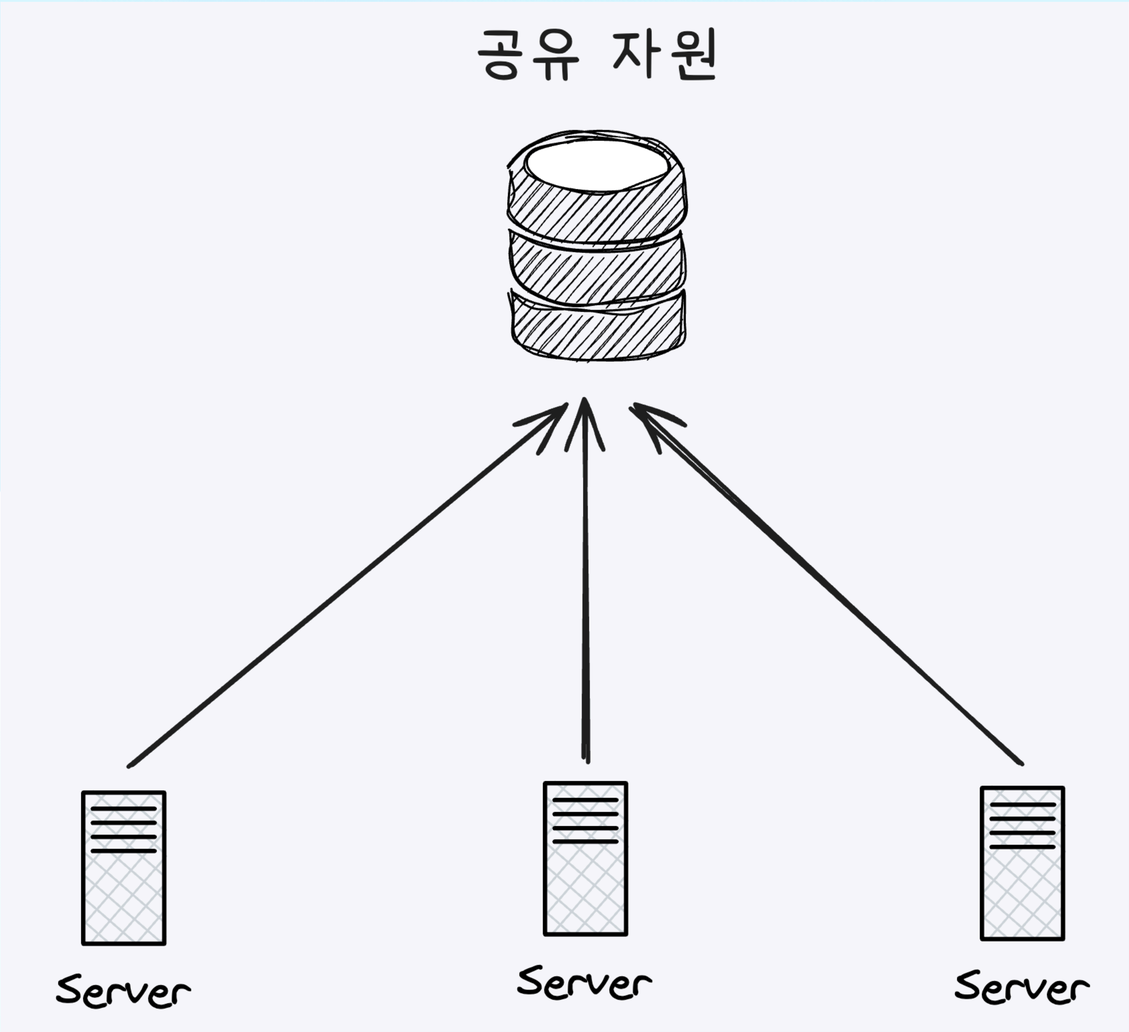
각자 스레드 존재 각자 스레드 존재 각자 스레드 존재

서버가 여러대면?

분산락이란?


AI 개요
분산락은 분산 시스템에서 여러 프로세스나 서버가 동시에 하나의 공유 자원에 접근하려고 할 때, 충돌을 방지하고 자원의 일관성을 유지하기 위해 사용되는 기술입니다. 특정 시점에 하나의 프로세스만 해당 자원에 접근하도록 제어하며, 이를 통해 시스템의 안정성을 높이고 데이터 무결성을 보장하는 것을 목표로 합니다.

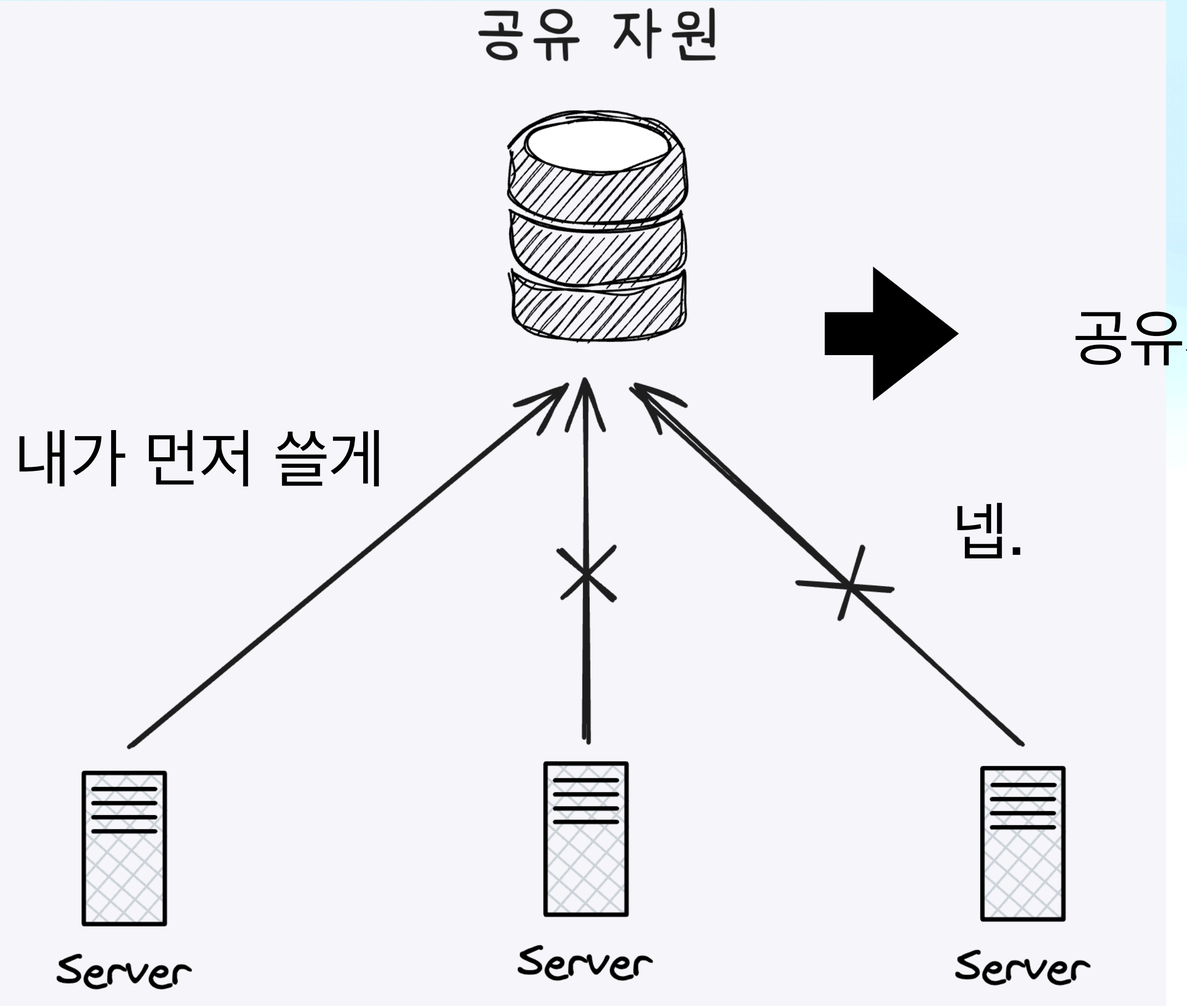
분산락이란?



각 서버가 서로 인식할 수 있는 락이 필요함

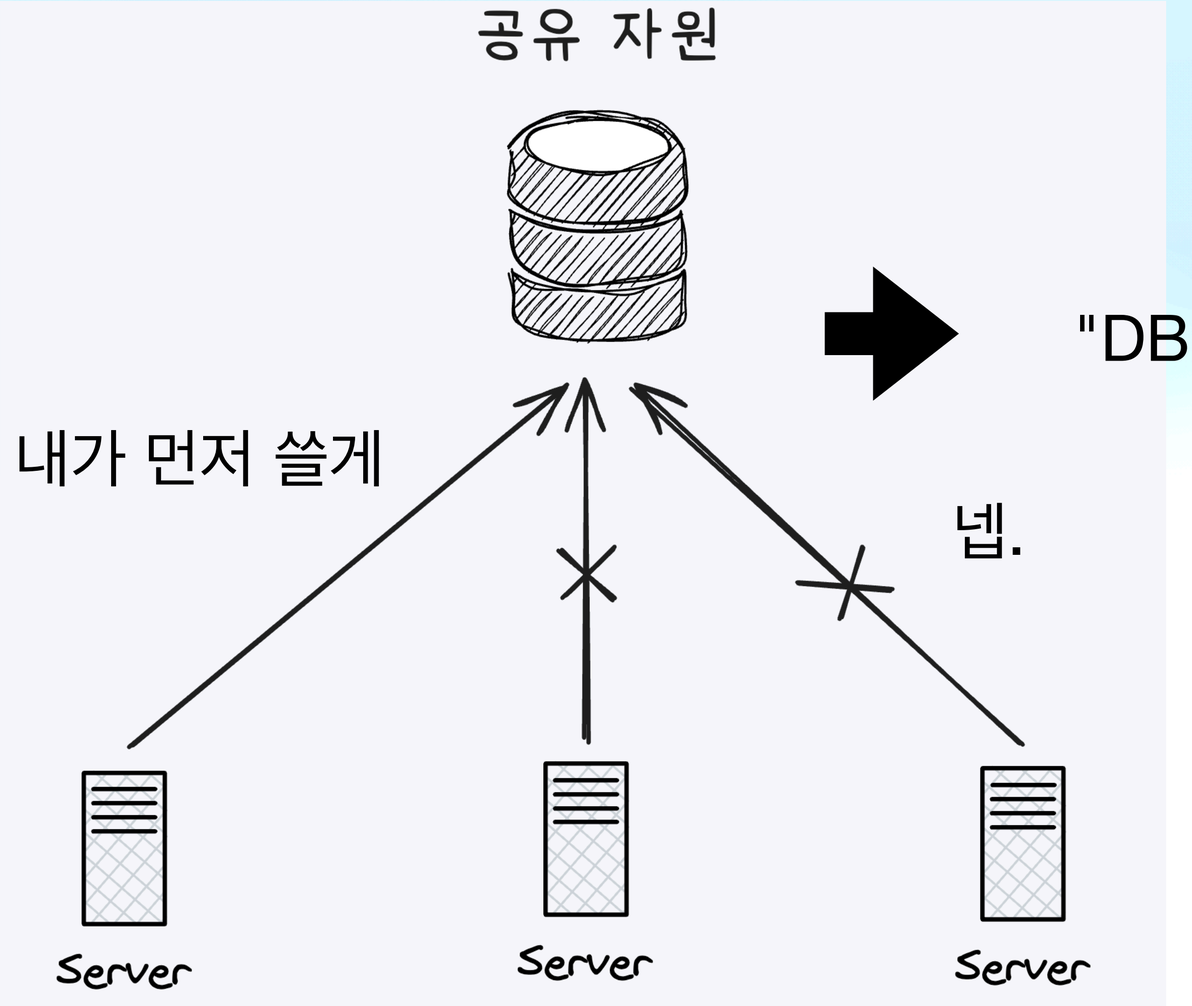
=> 공유 자원을 사용하면 되지않을까?

분산락이란?



공유자원에 락을 걸어서 한개의 요청만 허용한다

DB 분산락



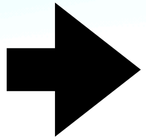
"DB"에 락을 걸어서 한개의 요청만 허용한다

DB 분산락

```
CREATE TABLE distributed_locks (
    lock_name VARCHAR(255) PRIMARY KEY
);

INSERT INTO distributed_locks (lock_name) VALUES ('my_resource_lock');
```

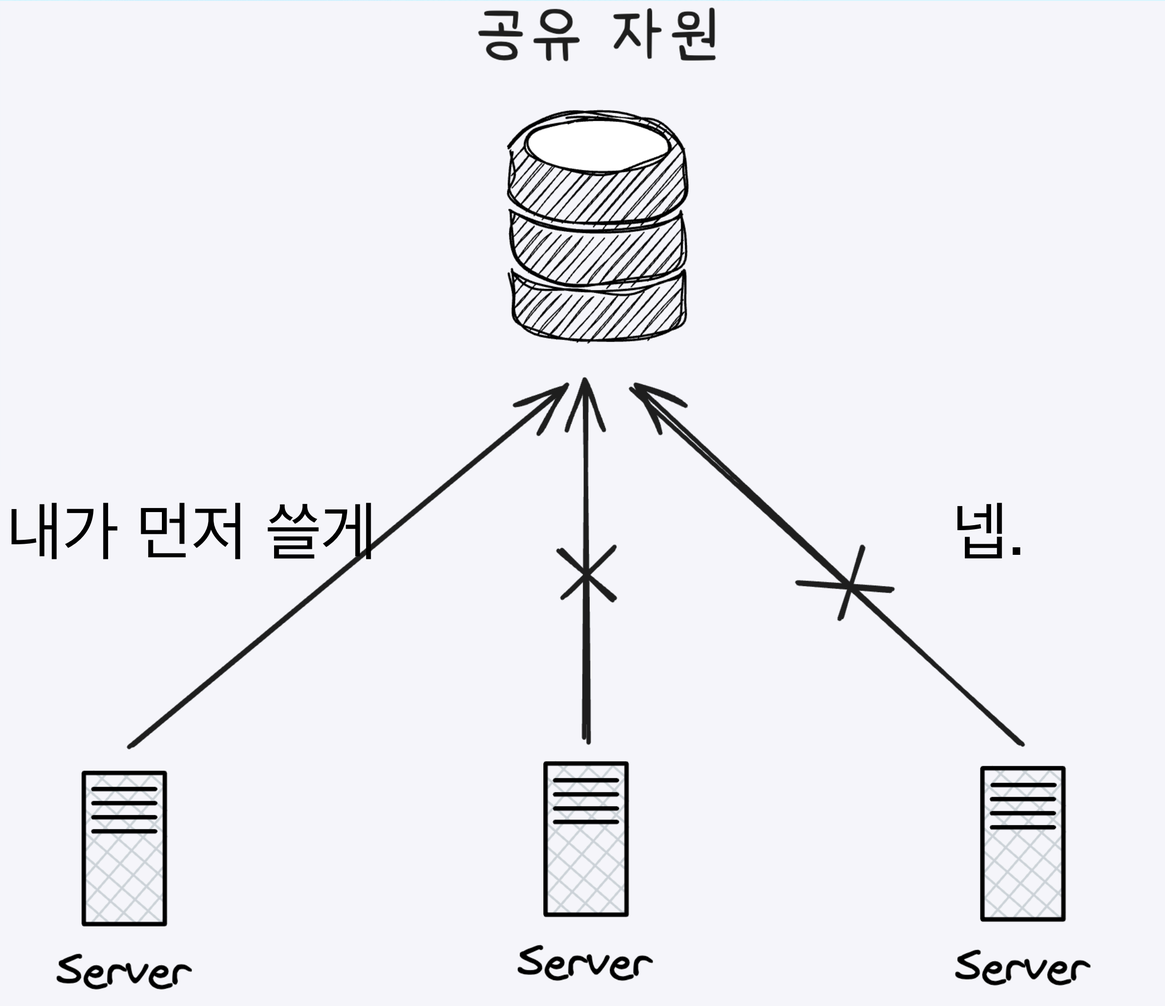
```
insert distribute_locks (lock_name) VALUES ('til_insert')
```

락용 테이블을 만든다

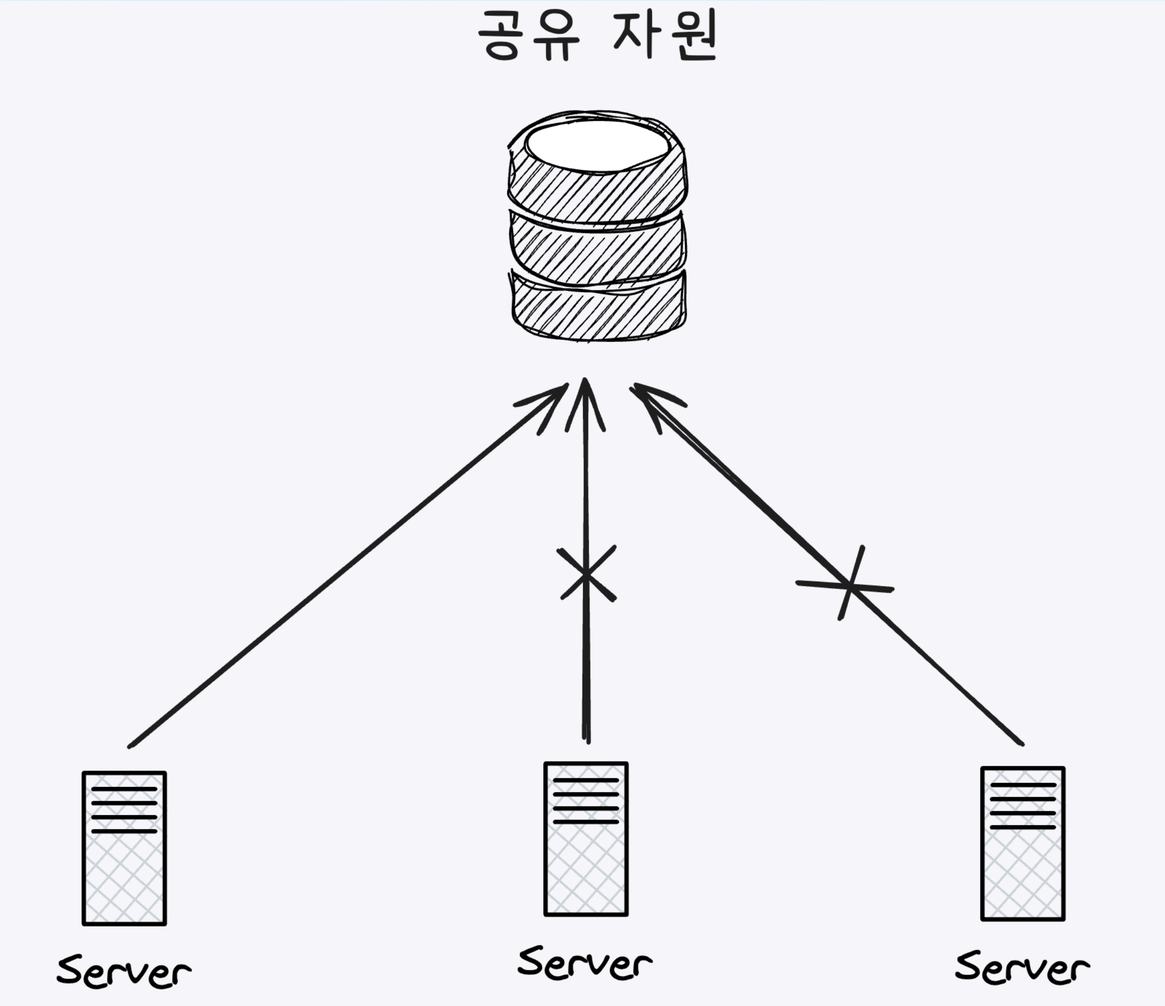


테이블에 락을 건다

DB 분산락 문제점

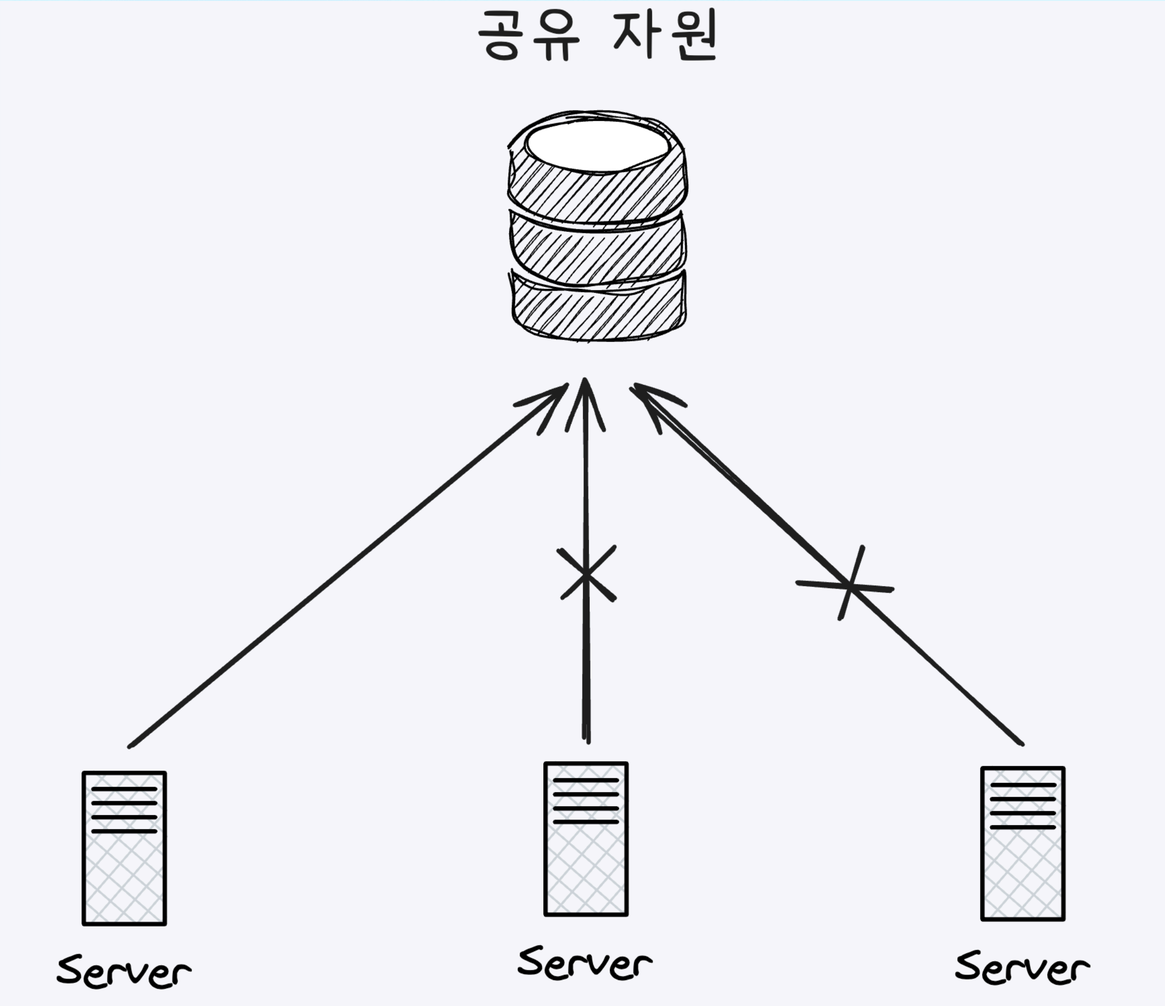


DB 분산락 문제점



락을 걸고 풀어주지 않으면? (장애등)

DB 분산락 문제점



락을 걸고 풀어주지 않으면? (장애등)

=> 락 타임아웃을 걸어야함

=> 락 관리를 해주어야한다

DB 분산락을 사용하는 라이브러리

(스케줄링 최적화)

QUARTZ

Job Scheduler

(좀 더 복잡한 스케줄링 최적화)

라이브러리를 잘 사용하면 락을 관리하지 않아도됨

shed lock 도입

```java
@Configuration  jeyong +1
@EnableScheduling
@EnableSchedulerLock(defaultLockAtMostFor = "5m")
public class SchedulingConfig implements SchedulingConfigurer {

    @Override  1 usage  jeyong
    public void configureTasks(final ScheduledTaskRegistrar taskRegistrar) {
        taskRegistrar.setTaskScheduler(taskScheduler());
    }

    // TODO. 추후 ThreadPoolTaskScheduler의 thread 설정 변경 및 graceful shutdown 고려 필요
    @Bean  jeyong
    public ThreadPoolTaskScheduler taskScheduler() {
        ThreadPoolTaskScheduler scheduler = new ThreadPoolTaskScheduler();
        scheduler.setThreadNamePrefix("scheduler-");

        return scheduler;
    }

    @Bean  SeungYeon
    public LockProvider lockProvider(final DataSource dataSource) {
        return new JdbcTemplateLockProvider(
                JdbcTemplateLockProvider.Configuration.builder()
                        .withJdbcTemplate(new JdbcTemplate(dataSource))
                        .usingDbTime()
                        .build()
        );
    }
}
```

auto configuration 설정

shed lock 도입

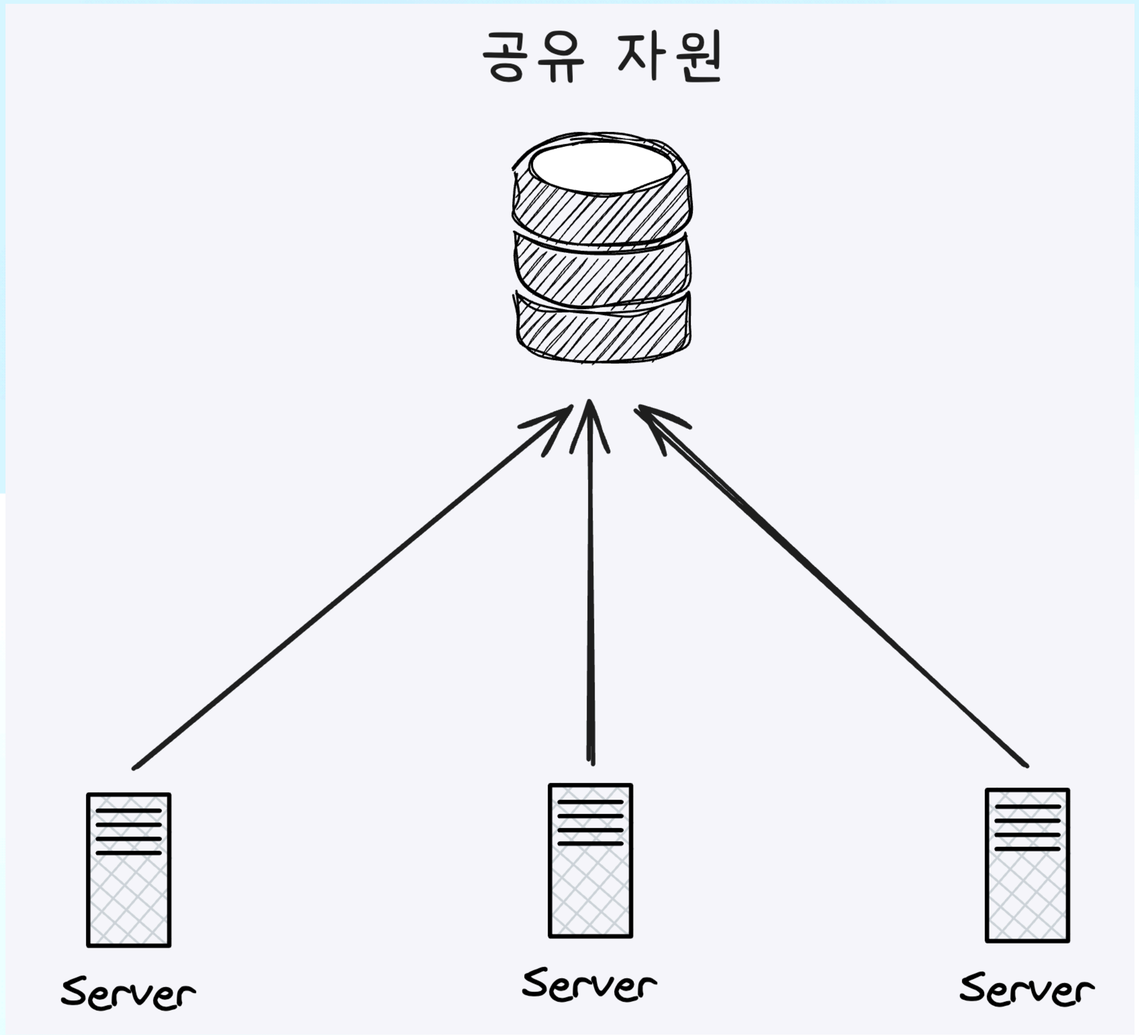
V202509161713__add_shedlock_table.sql

```sql
CREATE TABLE shedlock
(
    name       VARCHAR(64)  NOT NULL,
    lock_until TIMESTAMP(3) NOT NULL,
    locked_at  TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    locked_by  VARCHAR(255) NOT NULL,
    PRIMARY KEY (name)
);
```

테이블을 생성해야함!

```java
@Scheduled(cron = "0 */3 * * * *") every 3 minutes jeyong +1
@SchedulerLock(
        name = "notifyRegistrationClosingIn30Minutes",
        lockAtMostFor = "2m",
        lockAtLeastFor = "1m"
)
@Transactional
public void notifyRegistrationClosingIn30Minutes() {
    LocalDateTime now = LocalDateTime.now();
    LocalDateTime windowStart = now.plus(REGISTRATION_CLOSING_REMI
    LocalDateTime windowEnd = windowStart.plus(SCHEDULER_SCAN_WIND

    List<Event> upcomingEvents =
            eventRepository.findAllByEventOperationPeriodRegistrat

    upcomingEvents.stream()
            .filter( Event event -> !event.isFull()).
            forEach( Event upcomingEvent -> {
                List<OrganizationMember> recipients = getOptInNonG
                String content = "이벤트 신청 마감이 임박했습니다.

                sendAndRecordReminder(upcomingEvent, recipients, c
            });
}
```

shed lock 도입



알람을 한 서버만 보내게 해야했음

shed lock 도입

V202509161713__add_shedlock_table.sql

```sql
CREATE TABLE shedlock
(
    name       VARCHAR(64)  NOT NULL,
    lock_until TIMESTAMP(3) NOT NULL,
    locked_at  TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    locked_by  VARCHAR(255) NOT NULL,
    PRIMARY KEY (name)
);
```

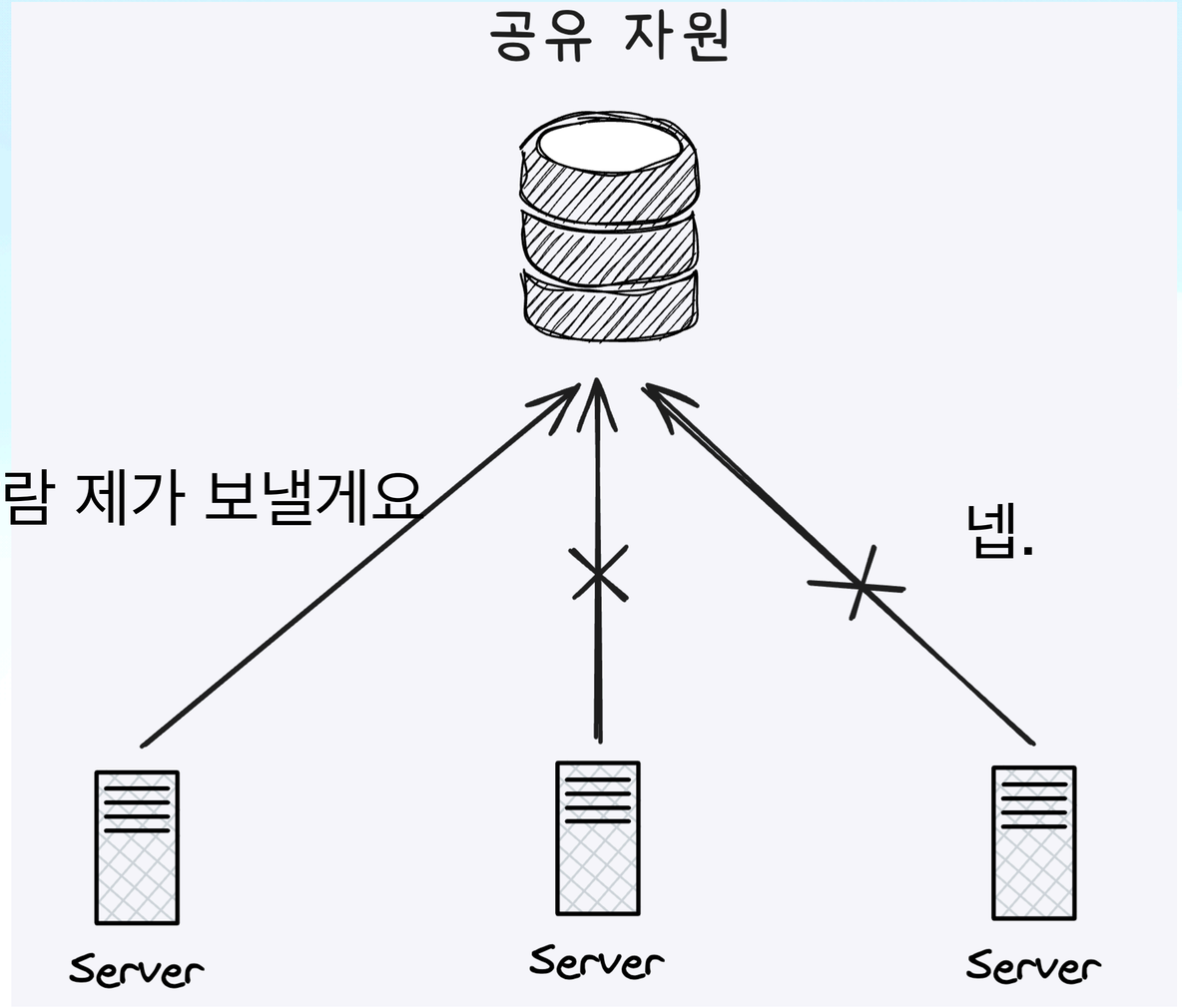
테이블을 생성해야함!

```java
@Scheduled(cron = "0 */3 * * * *") every 3 minutes jeyong +1
@SchedulerLock(
        name = "notifyRegistrationClosingIn30Minutes",
        lockAtMostFor = "2m",
        lockAtLeastFor = "1m"
)
@Transactional
public void notifyRegistrationClosingIn30Minutes() {
    LocalDateTime now = LocalDateTime.now();
    LocalDateTime windowStart = now.plus(REGISTRATION_CLOSING_REMI
    LocalDateTime windowEnd = windowStart.plus(SCHEDULER_SCAN_WIND

    List<Event> upcomingEvents =
            eventRepository.findAllByEventOperationPeriodRegistrat

    upcomingEvents.stream()
            .filter( Event event -> !event.isFull()).
            forEach( Event upcomingEvent -> {
                List<OrganizationMember> recipients = getOptInNonG
                String content = "이벤트 신청 마감이 임박했습니다.

                sendAndRecordReminder(upcomingEvent, recipients, c
            });
}
```

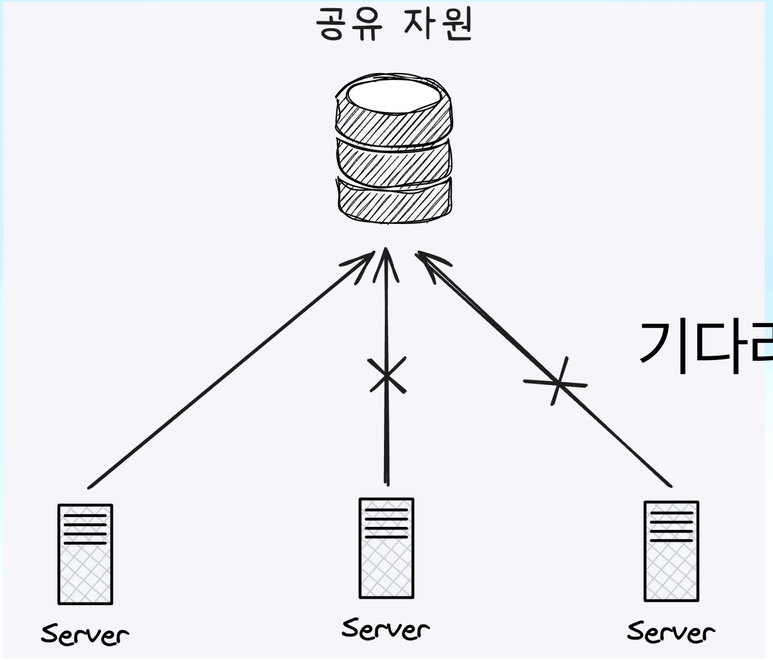
칼럼 이름을 지정해줘야함!

SchedulerLock 부분

shed lock 도입



구조적인 DB 분산락 문제점



기다려야지~~

락 외에도 **조회, 쓰기** 등 DB는 많은 트래픽을 받는다.

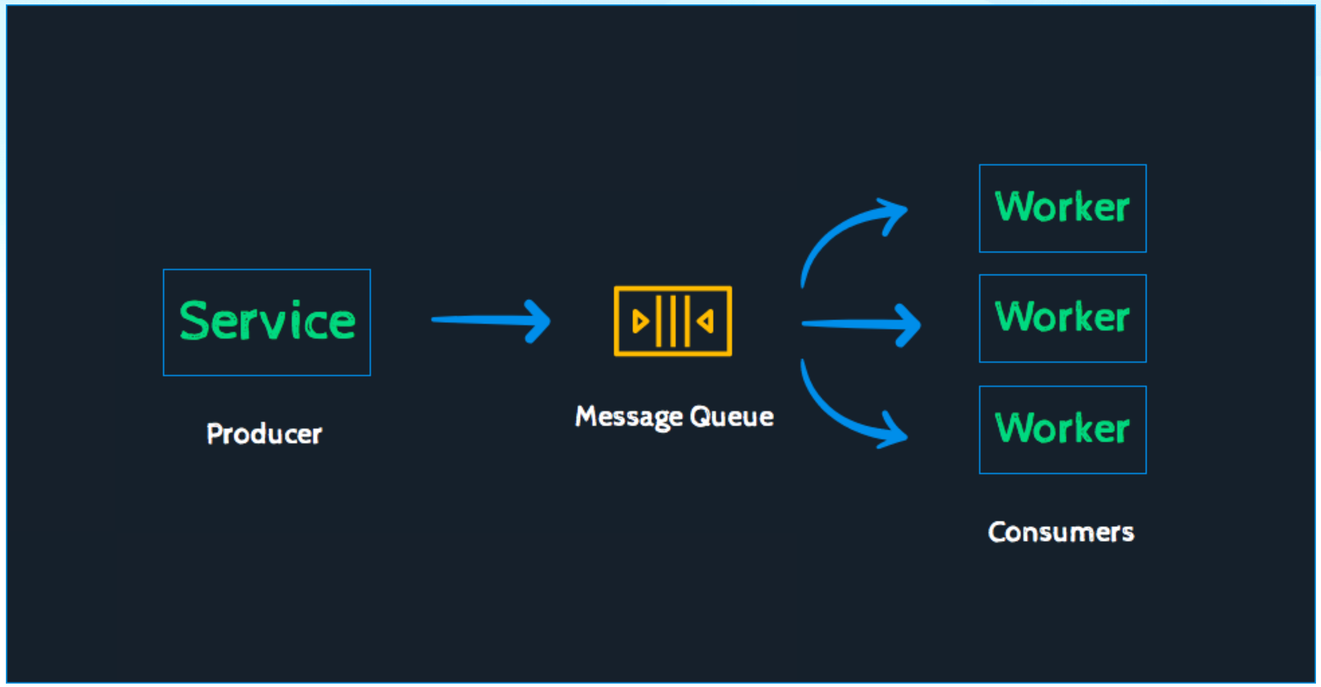
락을 빠르게 놓지 못할 수 있다

=> 락을 기다리는 병목이 생긴다

추가 아키텍쳐 사용

순차적으로 처리되는 기술들을 사용하면 된다.

싱글스레드 + 이벤트루프 방식으로 빠르면서도 동시성 문제가 발생하지 않는다



메시지큐 (kafka, rabbitmq으로 구현)

고민해보면 좋은 내용

락 경합이 없다고 무조건 메시지큐 아키텍쳐를 도입해도 되는가?

DB 분산락으로는 부족한가(왜 spring quertz 등에서는 db분산락을 쓰지?)

문제의 중요도, 우리 서비스에서 정의한 목표는 무엇인가

감사합니다