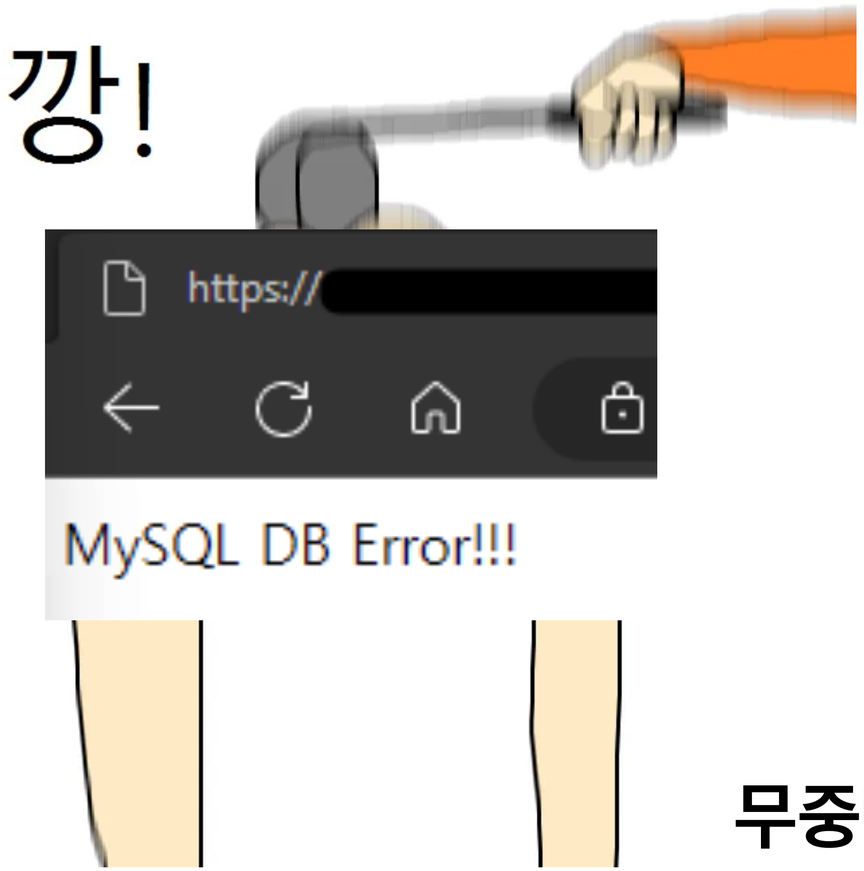
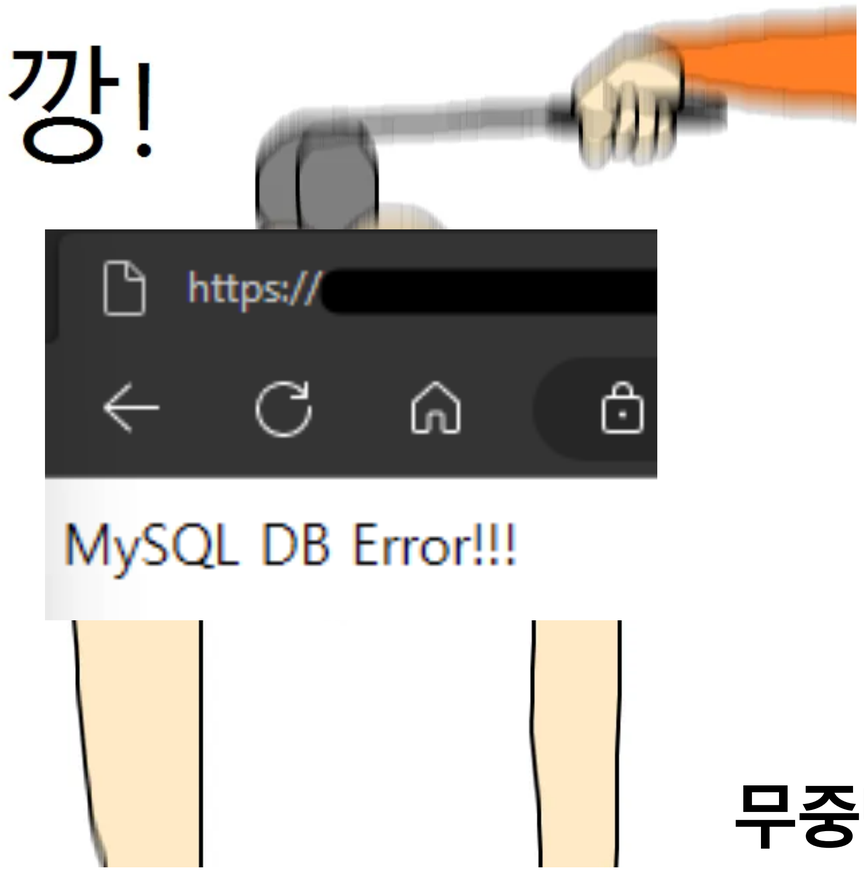


무중단 스키마 변경 2탄

서비스 중단 없이 DB 스키마 변경하기

백엔드 7기 모코



무중단 스키마 변경 2탄

서비스 중단 없이 DB 스키마 변경하기

백엔드 7기 모코

결론

그냥 중단 변경 하자. 무슨 무중단이여

- Offline DDL의 Online Migration은 여러 단점이 있다.
  - **DB의 부하**를 일으킨다. (성능, 용량)
  - **롤백**이 번거롭다.
- 무중단이라고 무조건 좋은 게 아니다.
  - 위의 사이드이펙트를 감수하고 할 정도로 시급한 게 아니라면
    그냥 **점검 시간**(Maintenance Window)을 가지고 **Offline Migration을 진행**하자.

결론

그냥 중단 변경 하자. 무슨 무중단이여

- Offline DDL의 Online Migration은 여러 단점이 있다.
  - **DB의 부하**를 일으킨다. (성능, 용량)
  - **롤백**이 번거롭다.
- 무중단이라고 무조건 좋은 게 아니다.
  - 위의 사이드이펙트를 감수하고 할 정도로 시급한 게 아니라면
    그냥 **점검 시간**(Maintenance Window)을 가지고 **Offline Migration을 진행**하자.

대규모 서비스를 운영 중인데 작은 스키마 변경이 필요하다면 전체 서비스를 중단해야 하나요?
잠깐의 중단이 큰 손실로 이어지는 서비스에서는 어떡해요?

카카오톡이 새벽 4시에 점검..?

서비스 중단 없이

안전하게 스키마를 변경할 방법은 없을까?

스키마 변경이 서비스를 중단시키는 이유

1. 데이터베이스 계층의 문제: 테이블 락

- DDL 실행 중 테이블에 **Read Lock이 걸려 읽기/쓰기가 차단**된다
- 대용량 테이블일수록 락 시간이 길어진다
- ex) 1억 row 테이블의 컬럼 수정 시 수십 분간 락 발생

2. 애플리케이션 계층의 문제: 스키마 불일치

- DDL 실행 시점과 애플리케이션 배포 시점의 시간차
- **변경된 테이블 구조와 기존 애플리케이션 코드의 불일치**
- ex) 컬럼 변경 시 기존 애플리케이션 엔티티 구조 mismatch

데이터베이스 계층의 문제: 테이블 락

https://www.youtube.com/watch?v=8ucMsqtZ_e8&t=11s

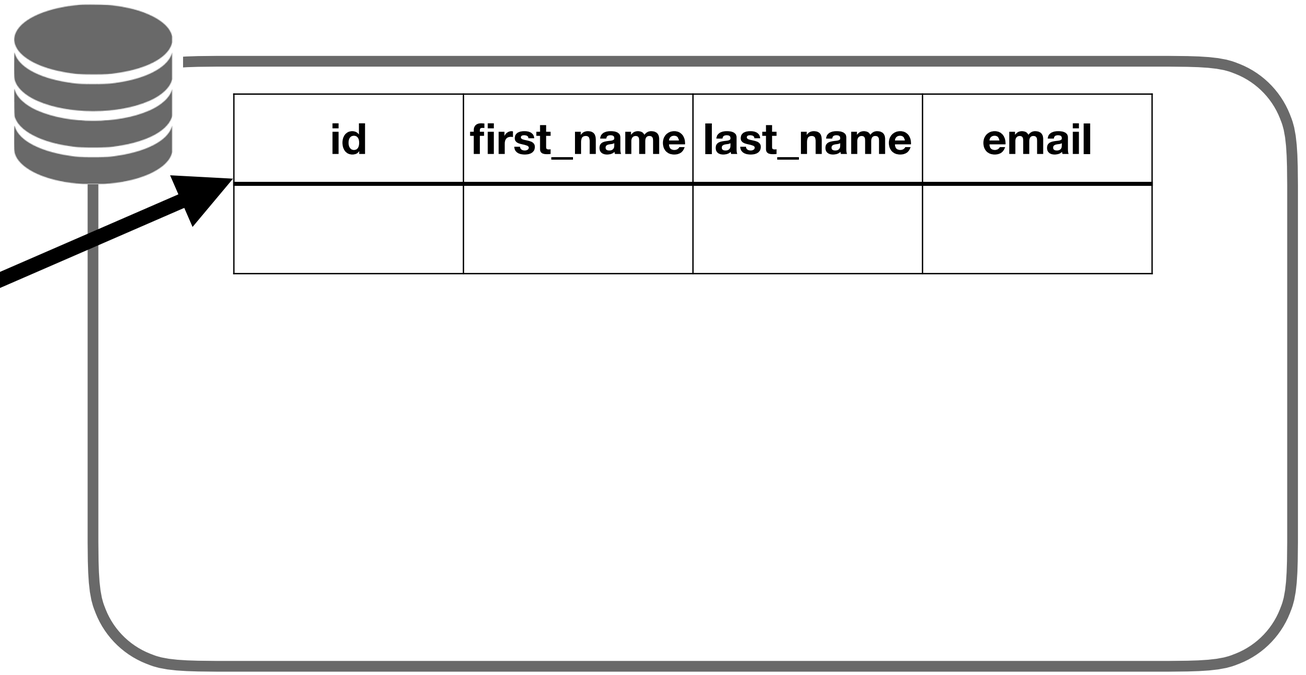
데이터베이스 계층의 문제: 테이블 락

요약

- DDL에는 Online DDL과 Offline DDL이 있다
  - Online DDL은 작업 간 **DML을 허용**한다.
  - Offline DDL은 테이블 락이 걸려 **DML이 차단**된다.(운영 상황에서 치명적)
- DDL 실행 전에는 조심하자
  - 공식 문서를 확인하여 사용되는 알고리즘이나 락 종류를 확인한다.
  - 로컬/개발 환경에서 알고리즘과 락 종류를 명시하여 실행(**테스트**)해보자.

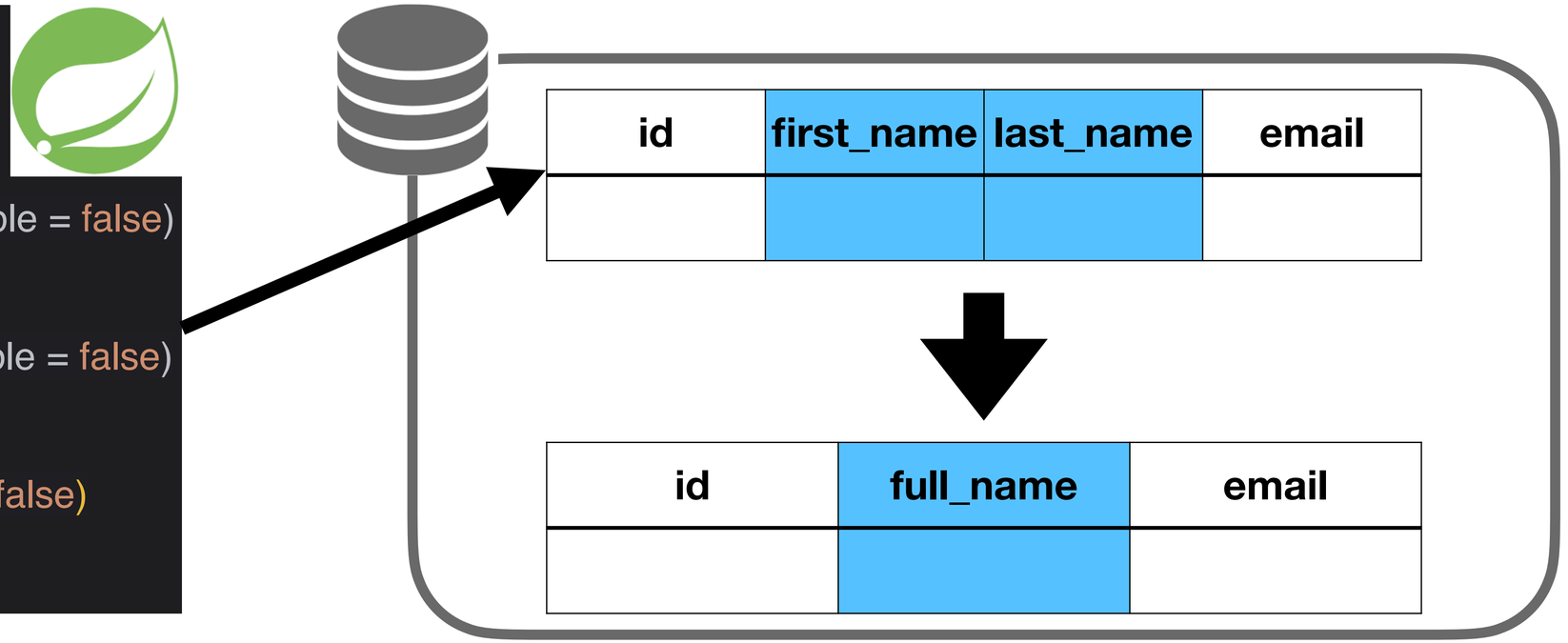
애플리케이션 계층의 문제: 스키마 불일치

```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "email", nullable = false)
    private String email;
}
```



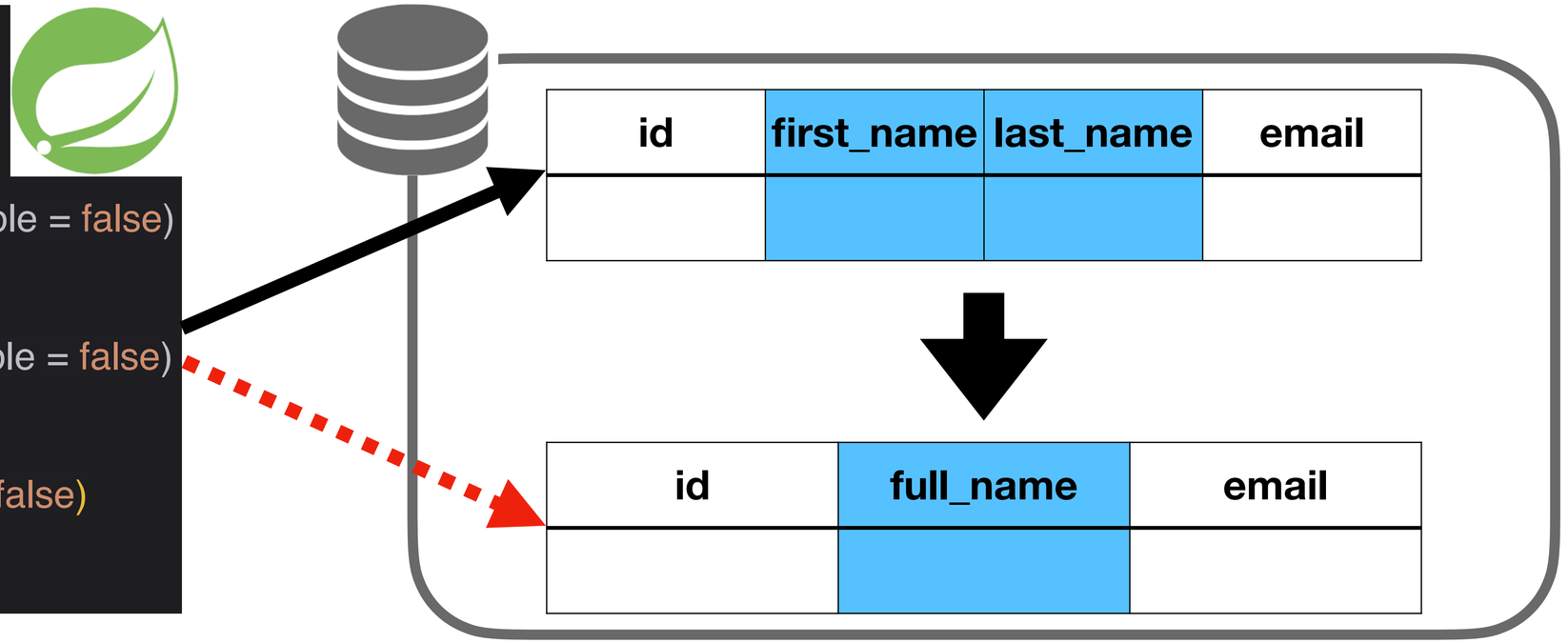
애플리케이션 계층의 문제: 스키마 불일치

```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "email", nullable = false)
    private String email;
}
```



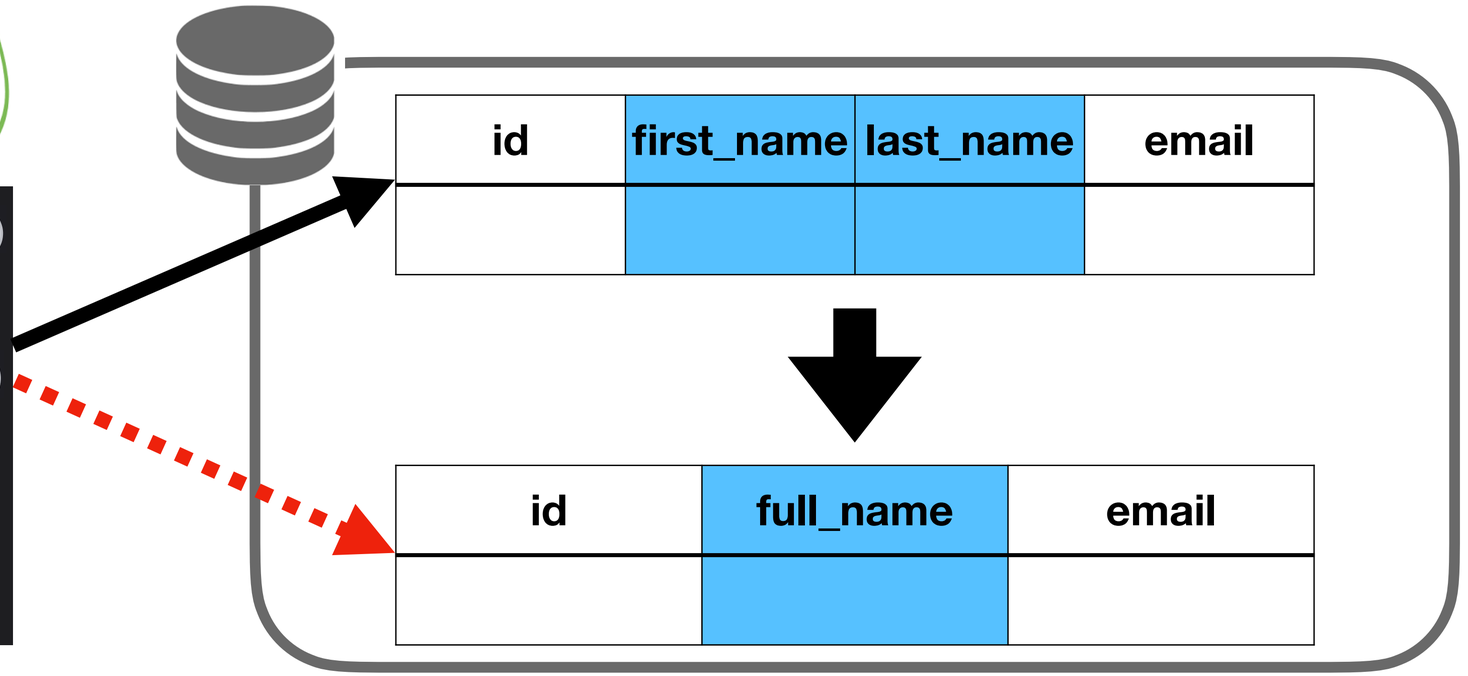
애플리케이션 계층의 문제: 스키마 불일치

```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "email", nullable = false)
    private String email;
}
```



애플리케이션 계층의 문제: 스키마 불일치

```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "email", nullable = false)
    private String email;
}
```



스키마 변경이 Online이든 Offline이든

애플리케이션은 스키마 변경 전후를 모두 처리할 수 있어야 한다.

Expand Contract Pattern

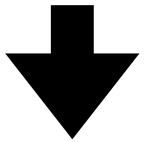
확장 축소 패턴

Expand Contract Pattern

확장 축소 패턴

id	first_name	last_name	email

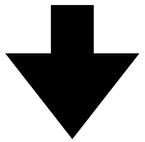
Expand(확장)



+ full_name

id	full_name	first_name	last_name	email

Contract(축소)



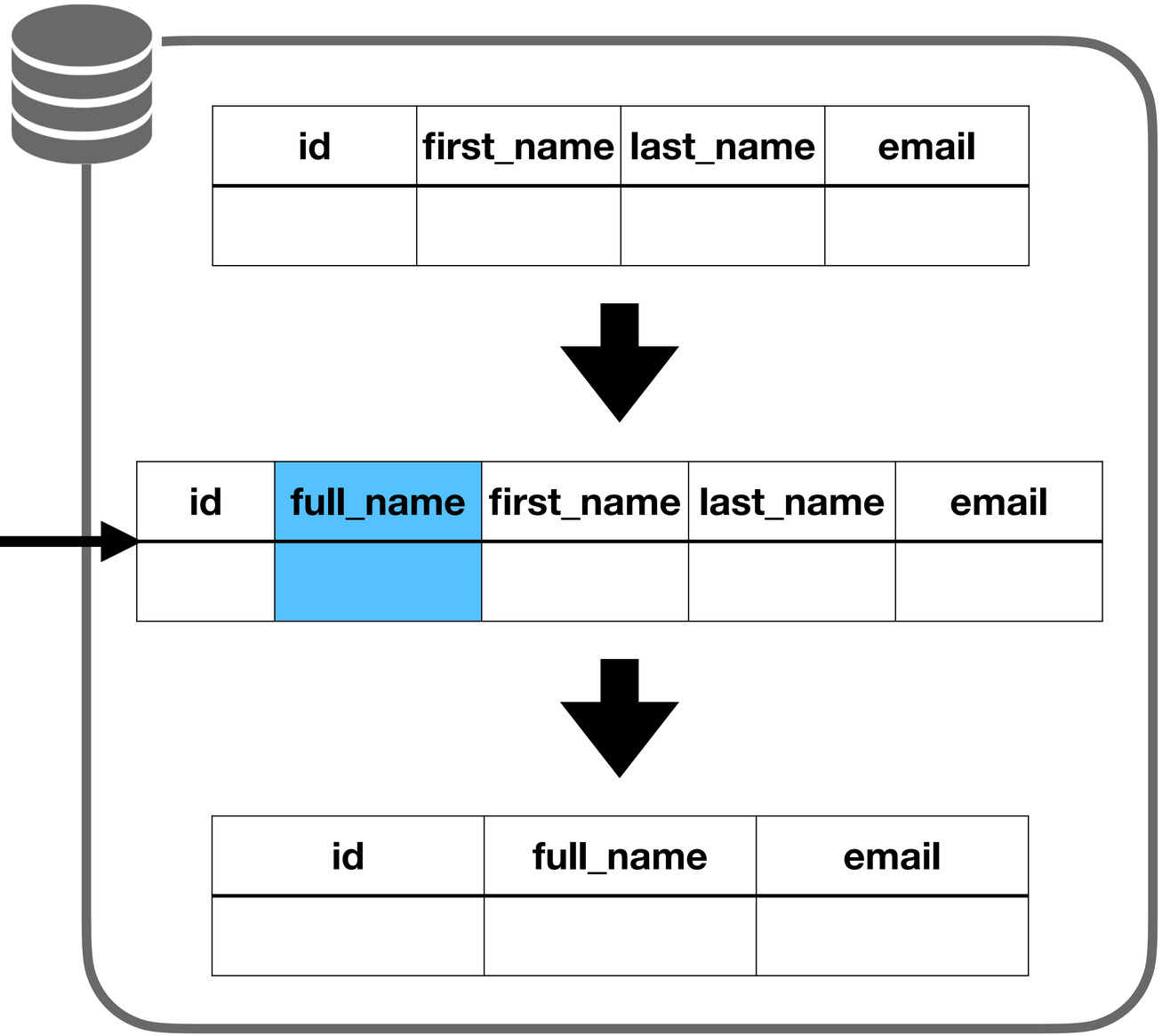
- first_name, last_name

id	full_name	email

Expand Contract Pattern

확장 축소 패턴

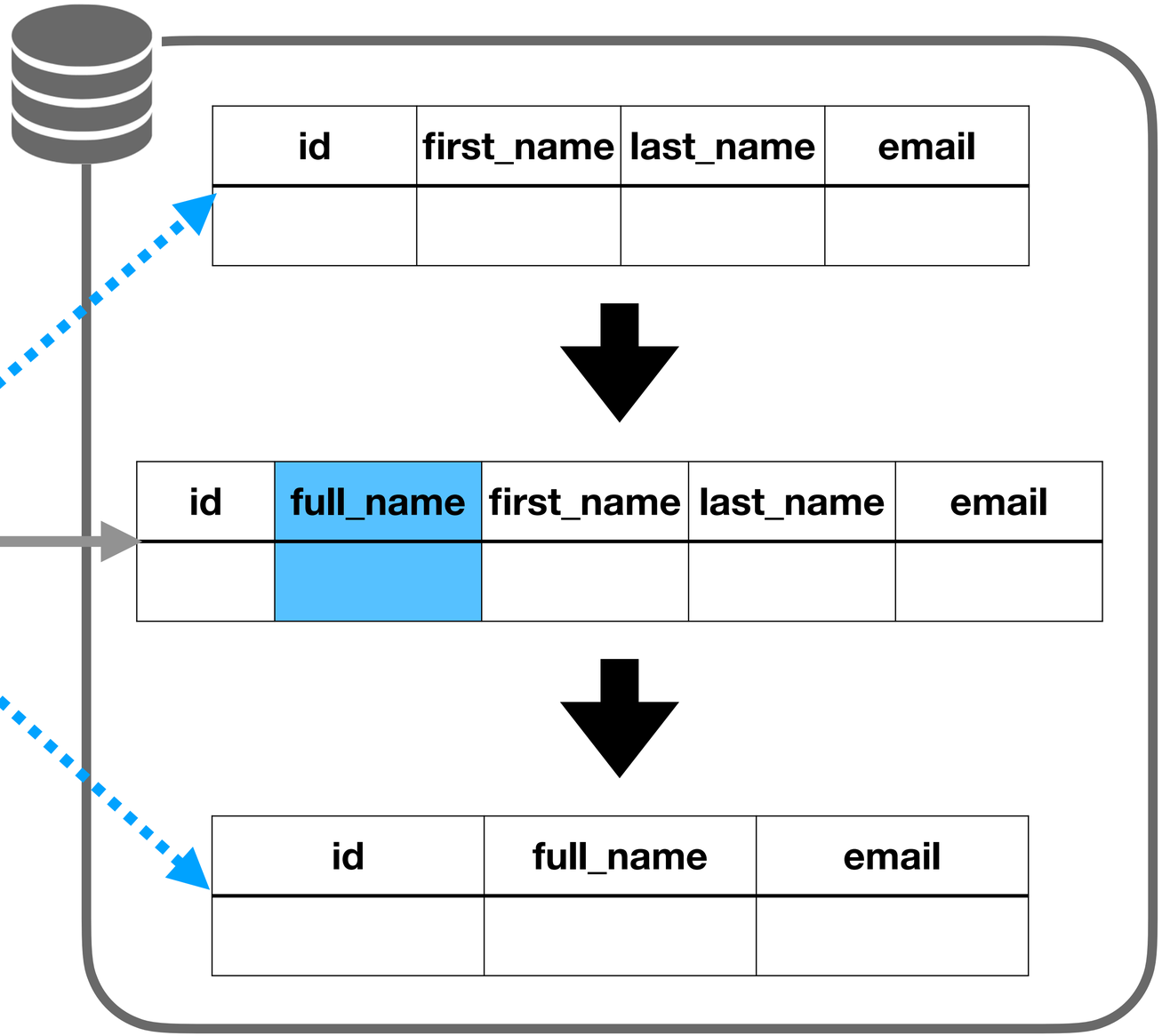
```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "full_name")
    private String fullName;

    @Column(name = "email", nullable = false)
    private String email;
}
```



Expand Contract Pattern

확장 축소 패턴

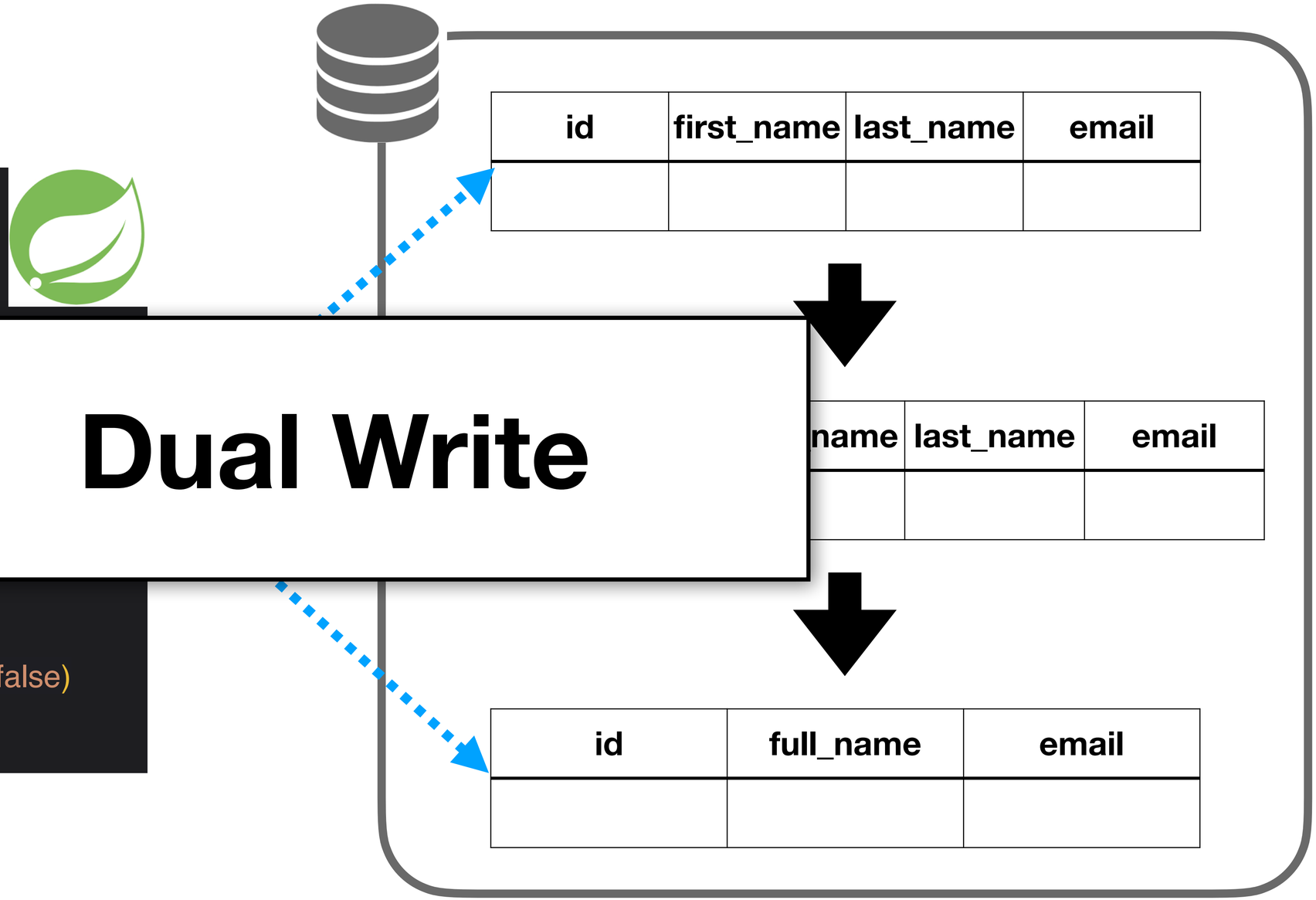
```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "full_name")
    private String fullName;

    @Column(name = "email", nullable = false)
    private String email;
}
```



Expand Contract Pattern

확장 축소 패턴

```
public class User {
    @Id
    private Long id;

    @Column(name = "first_nar
    private String firstName;

    @Column(name = "last_nar
    private String lastName;

    @Column(name = "full_nam
    private String fullName;

    @Column(name = "email", nullable = false)
    private String email;
}
```

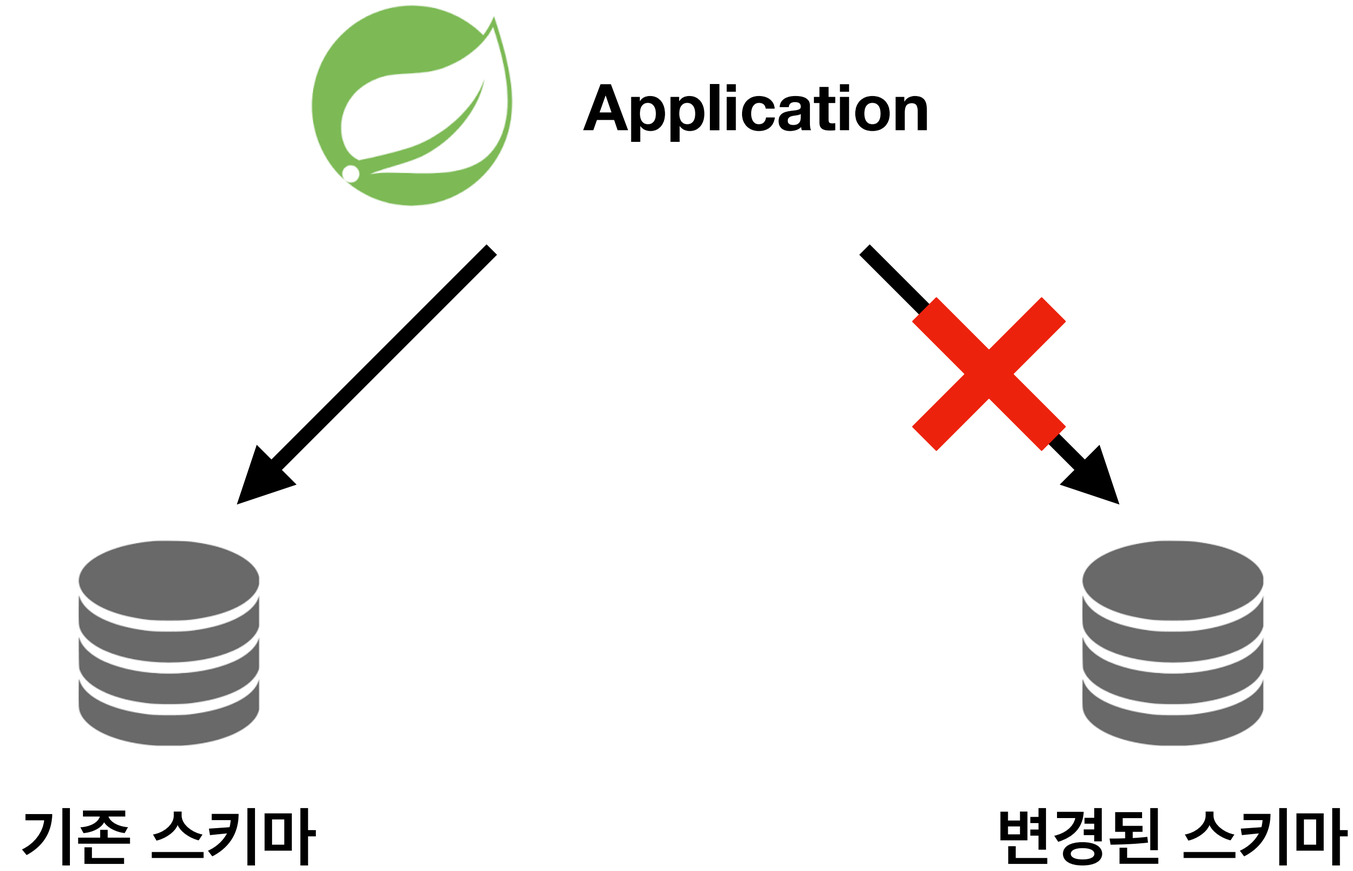


Dual Write

```java
public class User {
    @Id
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "full_name")
    private String fullName;

    @Column(name = "email", nullable = false)
    private String email;

    public User(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.fullName = firstName + " " + lastName;
        this.email = email;
    }

    public void updateName(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.fullName = firstName + " " + lastName;
    }
}
```

■ 기존 스키마 ■ 변경 스키마

Dual Write



괜찮네..

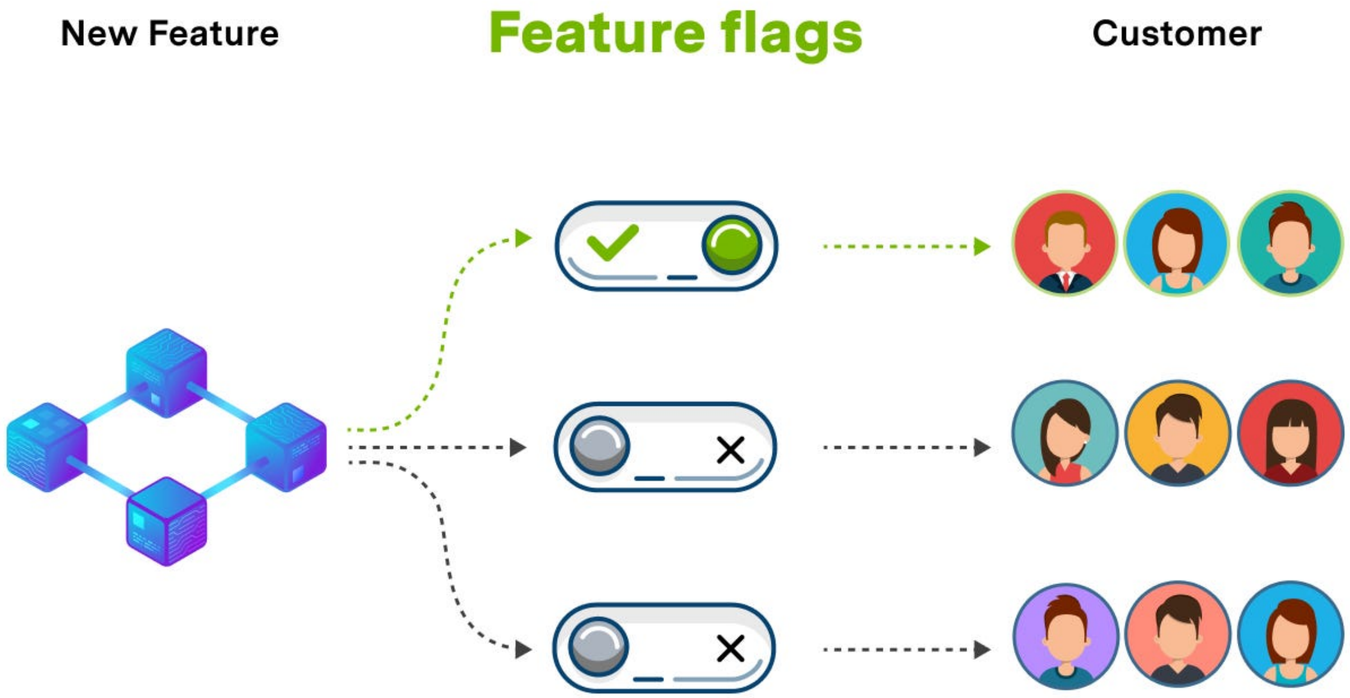
근데 Dual Write가 끝나고

애플리케이션이 변경된 스키마를 읽었는데 에러나면?

한번에 모든 사용자가 버그나는 거 아니야?

Feature Flag

Feature Flag



읽기 비율 1% -> 10% -> 50% -> 100% ...

점진적 전환

어? 나 그거 알아.

카나리 배포잖아!

Feature Flag
카나리 배포와의 차이

카나리 배포: **인프라 계층**에서의 점진적 전환

- 특정 조건에 대해 여러 요청을 보낼 시 **매번 서로 다른 WAS로 요청**이 전달됨

Feature Flag: **DB/애플리케이션 계층**에서의 점진적 전환

- 데이터 식별자를 기준으로 flag를 세우기 때문에 여러 요청에도 **일관적인 응답 반환**

사용자 경험의 차이

정리

서비스 중단 없이 DB 스키마 변경하기

Expand

스키마 변경 대상 추가

- 새 컬럼/테이블을 **추가만** 하고 기존은 유지
- 신규 데이터는 NULL 상태
- 애플리케이션은 아직 기존 스키마 사용

```
ALTER TABLE users ADD COLUMN email VARCHAR(255)
```

서비스 중단 없이 DB 스키마 변경하기

Expand

Dual Write

변경 전/후 쓰기 대응 애플리케이션 배포

- 신규/기존 구조 **양쪽에 모두 쓰기**
- 읽기는 아직 기존 스키마 사용
- 신규 데이터는 양쪽에 저장됨

```
public void updateName(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = firstName + " " + lastName;
}
```

서비스 중단 없이 DB 스키마 변경하기

Expand

Dual Write

BackFill

기존 데이터 마이그레이션

- 기존 데이터를 신규 스키마에 적용
- Spring Batch 등으로 **배치 처리**
- NULL -> 실제 값으로 채우기 (백필)

```
UPDATE users
    SET full_name = CONCAT(first_name, ' ', last_name)
    WHERE full_name IS NULL;
```

서비스 중단 없이 DB 스키마 변경하기

Expand

Dual Write

BackFill

Read Conversion

읽기 전환 애플리케이션 배포(선택)

- Feature Flag로 **신규 스키마 읽기 점진적 전환**
- 10% -> ... -> 100% 단계적 확대
- 쓰기는 여전히 Dual Write 유지

```
public String getDisplayName(boolean useNewSchema) {
    if (useNewSchema && fullName != null) {
        return fullName;
    }
    if (fullName == null) {
        log.warn("Full name is null, falling back to first and last name.");
    } else {
        return fullName;
    }
    return firstName + " " + lastName;
}
```

서비스 중단 없이 DB 스키마 변경하기

Expand

Dual Write

BackFill

Read Conversion

Clean Up

코드 정리 애플리케이션 배포

- Feature Flag 제거 (100% 전환 확정)
- Dual Write 제거 (신규 스키마만 Write)
- **기존 스키마 관련 코드 제거**

```
public String getFullName() {
    // if (fullName != null) {
    //     return fullName;
    // }
    // return firstName + " " + lastName;
    return fullName;
}
```

서비스 중단 없이 DB 스키마 변경하기

Expand

Dual Write

BackFill

Read Conversion

Clean Up

Contract

기존 스키마 대상 제거

- 기존 컬럼/테이블 **삭제**
- 애플리케이션이 더이상 기존 스키마를 참조하지 않음

```
ALTER TABLE users
    MODIFY COLUMN full_name VARCHAR(255) NOT NULL;
```

서비스 중단 없이 DB 스키마 변경하기

Expand	스키마 변경 대상 추가
Dual Write	변경 전/후 쓰기 대응 애플리케이션 배포
BackFill	기존 데이터 마이그레이션
Read Conversion	읽기 전환 애플리케이션 배포(선택)
Clean Up	코드 정리 애플리케이션 배포
Contract	기존 스키마 대상 제거

감사합니다