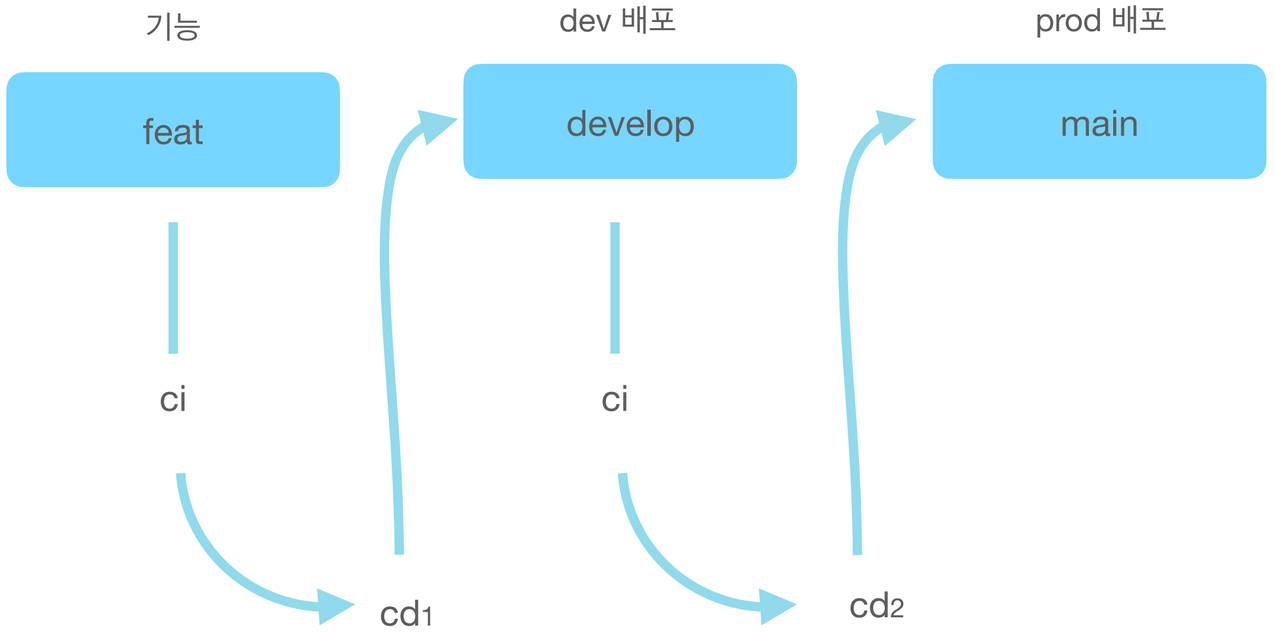
데이터 베이스 운영 / 안정성

BE 7기 칼리

목차



- ci: mysql 환경에서의 테스트 진행
- cd1: docker image 빌드 & 배포
- cd2: docker image 빌드 & 배포(브랜치명으로 버전 관리)

2. 버저닝

`git tag` 사용

`release` 브랜치에 커밋된 내역 중, 릴리즈 스코프 내에서 가장 마지막에 merge된 커밋에 태깅을 붙인다.

형식은 보통 **MAJOR.MINOR.PATCH** (예: 2.3.1) 으로 이루어집니다.

- **MAJOR (주 버전)**: 기존과 호환되지 않는 변경 발생 시 증가 (ex: API 파괴적 변경)
- **MINOR (부 버전)**: 기존과 호환되지만 새로운 기능이 추가될 때 증가
- **PATCH (수 버전)**: 버그 수정, 보안 패치 등 호환성을 깨지 않는 작은 수정

1. DROP X

2. Soft delete

1. DROP X

2. Soft delete

1. DROP X

1. DROP X

1. DROP X

1. DROP X

1. DROP X

1.1. 개념

- DDL 수준에서 DROP을 금지하여 데이터 보존성
- 실수/의도치 않은 스키마 손실 방지

1.2 정책

1.3. 구현

1. DROP X

1.1. 개념

- DDL 수준에서 DROP을 금지하여 데이터 보존성
- 실수/의도치 않은 스키마 손실 방지

1.2 정책

- 원칙: DROP 금지
- 예외: `-- ALLOW DROP` + 사유 기록(PR 번호 포함)
- 조건: 동일 컬럼/제약 변경 시 릴리즈 2회 이상 지난 경우에만 허용

```
-- ALLOW_DROP
-- reason: rename 과정에서 기존 제약/컬럼 drop 필요 (PR-1234)
ALTER TABLE notification
  DROP COLUMN recommended_target_amount;
```

1.3. 구현

1. DROP X

1.1. 개념

- DDL 수준에서 DROP을 금지하여 데이터 보존성
- 실수/의도치 않은 스키마 손실 방지

1.2 정책

- 원칙: DROP 금지
- 예외: `-- ALLOW DROP` + 사유 기록(PR 번호 포함)
- 조건: 동일 컬럼/제약 변경 시 릴리즈 2회 이상 지난 경우에만 허용

```
-- ALLOW_DROP
-- reason: rename 과정에서 기존 제약/컬럼 drop 필요 (PR-1234)
ALTER TABLE notification
  DROP COLUMN recommended_target_amount;
```

1.3. 구현

- Flyway 마이그레이션 CI 검사 로직
- PR 내 SQL 파일 변경 감지 후 DROP 탐지 및 실패 처리

1. DROP X

1.1. 개념

- DDL 수준에서 DROP을 금지하여 데이터 보존성
- 실수/의도치 않은 스키마 손실 방지

1.2 정책

- 원칙: DROP 금지
- 예외: `-- ALLOW DROP` + 사유 기록(PR 번호 포함)
- 조건: 동일 컬럼/제약 변경 시 릴리즈 2회 이상 지난 경우에만 허용

```
-- ALLOW_DROP
-- reason: rename 과정에서 기존 제약/컬럼 drop 필요 (PR-1234)
ALTER TABLE notification
  DROP COLUMN recommended_target_amount;
```

1.3. 구현

- Flyway 마이그레이션 CI 검사 로직
- PR 내 SQL 파일 변경 감지 후 DROP 탐지 및 실패 처리

1.3.1 CI 검사 로직

1.3.1.1. 적용 범위

- 경로: `backend/src/main/resources/db/migration/**.sql`
- 대상: PR 내 변경된 SQL 파일
- 실행 위치: develop-be → develop ci 검사 단계

1.3.1.2. CI 검사 로직 흐름 (1)

1. 변경 파일 목록 추출
2. ALLOW DROP 여부 확인
3. DROP 키워드 검사 후 위반 시 실패 처리

1. DROP X

2. Soft delete

1. DROP X

2. Soft delete

2. Soft delete

2. Soft delete

2. Soft delete

2. Soft delete

2. Soft delete

2. Soft delete

2. Soft delete

2.1. 개념

- DELETE 수행 시 실제 삭제가 아닌 deleted_at 갱신으로 처리
- 데이터 복구 및 이력 추적 가능

2.2. 스키마 규칙

2. Soft delete

2.1. 개념

- DELETE 수행 시 실제 삭제가 아닌 deleted_at 갱신으로 처리
- 데이터 복구 및 이력 추적 가능

2.2. 스키마 규칙

2.2.1. 공통 컬럼

- `created_at datetime NOT NULL DEFAULT CURRENT_TIMESTAMP;`
- `deleted_at datetime NULL`

2.2.2. 기본값의 의미

- `created_at` : INSERT 시 서버 시간이 자동 기록
- `deleted_at` : 소프트 삭제 시 서버 시간이 기록됨

2.2.3. JPA 매핑 규칙

2. Soft delete

2.1. 개념

- DELETE 수행 시 실제 삭제가 아닌 deleted_at 갱신으로 처리
- 데이터 복구 및 이력 추적 가능

2.2. 스키마 규칙

2.2.1. 공통 컬럼

- `created_at datetime NOT NULL DEFAULT CURRENT_TIMESTAMP;`
- `deleted_at datetime NULL`

2.2.2. 기본값의 의미

- `created_at` : INSERT 시 서버 시간이 자동 기록
- `deleted_at` : 소프트 삭제 시 서버 시간이 기록됨

2.2.3. JPA 매핑 규칙

```
@MappedSuperclass
public class BaseEntity {

    @Column(name = "created_at")
    protected LocalDateTime createdAt;

    @Column(name = "deleted_at")
    private LocalDateTime deletedAt;
}


@SQLRestriction("deleted_at IS NULL") // 항상 “활성”만 조회
@SQLDelete(sql="UPDATE cup_emoji SET deleted_at = NOW() WHERE id = ?")
public class CupEmoji extends BaseEntity {
    // ...
}
```

- @SQLRestriction("deleted_at IS NULL") : 기본 조회 필터 적용
- @SQLDelete: delete() 호출 시 soft delete로 변환

2. Soft delete

2.1. 개념

- DELETE 수행 시 실제 삭제가 아닌 deleted_at 갱신으로 처리
- 데이터 복구 및 이력 추적 가능

2.2. 스키마 규칙

2.2.1. 공통 컬럼

- `created_at datetime NOT NULL DEFAULT CURRENT_TIMESTAMP;`
- `deleted_at datetime NULL`

2.2.2. 기본값의 의미

- `created_at` : INSERT 시 서버 시간이 자동 기록
- `deleted_at` : 소프트 삭제 시 서버 시간이 기록됨

2.2.3. JPA 매핑 규칙

```
@MappedSuperclass
public class BaseEntity {

    @Column(name = "created_at")
    protected LocalDateTime createdAt;

    @Column(name = "deleted_at")
    private LocalDateTime deletedAt;
}


@SQLRestriction("deleted_at IS NULL") // 항상 “활성”만 조회
@SQLDelete(sql="UPDATE cup_emoji SET deleted_at = NOW() WHERE id = ?")
public class CupEmoji extends BaseEntity {
    // ...
}
```

- @SQLRestriction("deleted_at IS NULL") : 기본 조회 필터 적용
- @SQLDelete: delete() 호출 시 soft delete로 변환

2.2.3.1. 엔티티/테이블 예시

- cup_emoji 테이블 스키마 및 매핑 예시

```
mysql> select * from cup_emoji;
+----+-----------------------------------------------------------------------------------+---------------------+------------+
| id | url                                                                               | created_at          | deleted_at |
+----+-----------------------------------------------------------------------------------+---------------------+------------+
|  1 | https://github.com/user-attachments/assets/783767ab-ee37-4079-8e38-e08884a8de1c | 2025-08-01 10:10:10 | NULL       |
|  2 | https://github.com/user-attachments/assets/393fc8f9-bc46-4856-bfbe-889efc97151e | 2025-08-01 10:10:11 | NULL       |
+----+-----------------------------------------------------------------------------------+---------------------+------------+
2 rows in set (0.01 sec)
```

2. Soft delete

2.3. member 테이블에서의 active nickname 필드

2.3.1. 목적 배경

2.3.2. 기존 테이블에서의 문제 상황 시나리오

2. Soft delete

2.3. member 테이블에서의 active nickname 필드

2.3.1. 목적 배경

기존 member 테이블 구성으로 닉네임의 유니크 제약 조건을 검증할 수 있는 방법이 없었음

2.3.2. 기존 테이블에서의 문제 상황 시나리오

2. Soft delete

2.3. member 테이블에서의 active nickname 필드

2.3.1. 목적 배경

기존 member 테이블 구성으로 닉네임의 유니크 제약 조건을 검증할 수 있는 방법이 없었음

2.3.2. 기존 테이블에서의 문제 상황 시나리오

2.1.3.1. 기존 member 테이블 구성

```sql
CREATE TABLE member
(
    id                                BIGINT PRIMARY KEY AUTO_INCREMENT,
    nickname                          VARCHAR(10)  NOT NULL UNIQUE,
    gender                            VARCHAR(255),
    weight                            DOUBLE,
    target_amount                     INT          NOT NULL,
    is_marketing_notification_agreed  BOOLEAN      NOT NULL DEFAULT FALSE,
    is_night_notification_agreed      BOOLEAN      NOT NULL DEFAULT FALSE
);
```

- 멤버 nickname에 유니크 제약 조건 걸려 있음

2. Soft delete

2.1.3.2. 시나리오

1. 체체 닉네임을 가진 유저 1 생성

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	null

2. 유저1 탈퇴 → soft delete 적용

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00

3. 체체 닉네임을 가진 유저2 생성

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00
2	체체	2025-08-23T10:00:00	null

→ 문제 발생 : nickname 필드에 유니크 제약 조건이 걸려 있어서 불가능

2. Soft delete

2.1.3.2. 시나리오

1. 체체 닉네임을 가진 유저 1 생성

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	null

2. 유저1 탈퇴 → soft delete 적용

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00

3. 체체 닉네임을 가진 유저2 생성

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00
2	체체	2025-08-23T10:00:00	null

→ 문제 발생 : nickname 필드에 유니크 제약 조건이 걸려 있어서 불가능

2.3.3. 고민한 해결 방안 모색

1. `nickname` 필드와 `deleted_at` 복합 유니크 키로 수정

시나리오

1. 체체 닉네임을 가진 유저 1 생성

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	null

2. 체체 닉네임을 가진 유저 2 생성

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	null
2	체체	2025-08-23T10:00:00	null

→ 문제 발생 : duplicate 예외가 발생하지 않음

- nickname과 deleted_at 복합 시, 값이 같지 않다고 함
- mysql DB 상 null과 null은 같지 않은 값

2. Soft delete

결론 : active_nickname 필드 추가

1. 체체 닉네임을 가진 유저 1 생성

id	nickname	created_at	deleted_at	active_nickname
1	체체	2025-08-21T10:00:00	null	체체

2. 유저1 탈퇴 → soft delete 적용

id	nickname	created_at	deleted_at	active_nickname
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00	null

3. 체체 닉네임을 가진 유저2 생성

id	nickname	created_at	deleted_at	active_nickname
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00	null
2	체체	2025-08-23T10:00:00	null	체체

→ 성공 : nickname의 데이터를 삭제하지 않고도, 유니크 제약 조건을 검증할 수 있음

1. 스키마 무중단 변경

2. DB 권한 분리

1. 스키마 무중단 변경

2. DB 권한 분리

1. 스키마 무중단 변경

1. 스키마 무중단 변경

1. 스키마 무중단 변경

1. 스키마 무중단 변경

1.1. Expend / Contract 패턴

- Expend: 호환성 유지 단계
- Contract: 기존 스키마 의존 제거 단계

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1.4. 무중단 변경 특성

- 장시간 테이블 락 회피
- 시작/끝의 짧은 MDL 불가피

1. 스키마 무중단 변경

1.1. Expend / Contract 패턴

- Expend: 호환성 유지 단계
- Contract: 기존 스키마 의존 제거 단계

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1.4. 무중단 변경 특성

- 장시간 테이블 락 회피
- 시작/끝의 짧은 MDL 불가피

1. 스키마 무중단 변경

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1. 스키마 무중단 변경

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1. 스키마 무중단 변경

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1. 스키마 무중단 변경

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1.2.1. 단계별 스키마 변화

1.2.1.1. old 테이블

```
CREATE TABLE member_old (
  id BIGINT PRIMARY KEY,
  nickname VARCHAR(10) NOT NULL UNIQUE,
  created_at DATETIME NOT NULL,
  deleted_at DATETIME NULL
);
```

데이터 예시

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00
2	칼리	2025-08-23T10:00:00	null

1.2.1.2. new 테이블

```
CREATE TABLE member_new (
  id BIGINT PRIMARY KEY,
  nickname VARCHAR(10) NOT NULL,
  created_at DATETIME NOT NULL,
  deleted_at DATETIME NULL,
  active_nickname VARCHAR(10) -- 신규 컬럼
);
```

초기 백필

id	nickname	created_at	deleted_at	active_nickname
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00	null
2	칼리	2025-08-23T10:00:00	null	칼리

1. 스키마 무중단 변경

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1.2.1.3. Binlog 적용 시점

- 백필 후, 운영 중 신규 데이터 발생 → Binlog로 동기화

2025-08-24T10:00:00 ~ 2025-08-24T11:00:00 동안 new 테이블을 만들어서 데이터 옮기고 있음

Old 테이블

id	nickname	created_at	deleted_at
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00
2	칼리	2025-08-23T10:00:00	null
3	환노	2025-08-24T10:30:00	null

new 테이블 (binlog 적용 전)

id	nickname	created_at	deleted_at	active_nickname
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00	null
2	칼리	2025-08-23T10:00:00	null	칼리

Binlog 적용 후 (동기화 완료)

id	nickname	created_at	deleted_at	active_nickname
1	체체	2025-08-21T10:00:00	2025-08-22T10:00:00	null
2	칼리	2025-08-23T10:00:00	null	칼리
3	환노	2025-08-24T10:30:00	null	환노

1. 스키마 무중단 변경

1.2. gh-ost 원리

- ghost 테이블 생성 및 백필
- binlog 기반 변경 동기화
- 최종 RENAME TABLE 교체

1.2.1.4. 최종 안정화

- 애플리케이션 코드에서 new.active_nickname 컬럼 사용으로 전환
- Old 스키마 제거 (RENAME → DROP)
- 짧은 MDL 발생

```
DROP TABLE member_old_bak;
RENAME TABLE member_new TO member;
```

1. 스키마 무중단 변경

2. DB 권한 분리

2. DB 권한 분리

2. DB 권한 분리

2. DB 권한 분리

2. DB 권한 분리

2.3. DB 권한 규칙

1.3.1. 계정 분리 (2)

권한	유저 이름	설명
Read Only	mulkkam_ro	SELECT만 허용
Read Write	mulkkam_rw	DML 허용
Migrator	mulkkam_migrator	DDL + DML 모두 허용

1.3.2. 계정 생성 및 권한 부여 (3)

- /docker-entrypoint-initdb.d 폴더 내 초기화 SQL 스크립트 작성
- docker-compose volume으로 DB 기동 시 자동 적용

1.3.3. 애플리케이션 연동 (4)

- application-*.yml에서 각 권한별 계정 분리
- DataSourceConfig에서 ro, rw, DataSource Bean 등록
- ReadWriteRoutingDataSource로 트랜잭션 속성에 따라 동적 라우팅

1. 변경 관리

2. 복구 전략

1. 변경 관리

2. 복구 전략

1. 변경 관리

1. 변경 관리

1. 변경 관리

1.1. expend 스크립트

```
ALTER TABLE member
  ADD COLUMN active_nickname VARCHAR(10)
    GENERATED ALWAYS AS (CASE WHEN deleted_at IS NULL THEN nickname ELSE NULL END

CREATE UNIQUE INDEX uq_member_active_nickname ON member (active_nickname);
```

1.2. contract 스크립트

```
-- V20250821_223001__member_nickname_unique_contract.sql
-- nickname 컬럼에 걸린 고유 인덱스명을 동적으로 조회해 드롭
SET @idx := (
  SELECT index_name
  FROM information_schema.statistics
  WHERE table_schema = DATABASE()
    AND table_name   = 'member'
    AND column_name  = 'nickname'
    AND non_unique   = 0
    AND index_name  <> 'PRIMARY'
  LIMIT 1
);

SET @sql := IF(@idx IS NOT NULL,
               CONCAT('ALTER TABLE member DROP INDEX `', @idx, '`'),
               'SELECT 1');  -- 없으면 noop

PREPARE stmt FROM @sql;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

2. 복구 전략

2.1. 백업 덤프 시점으로 롤백

- 특정 시점에 찍어둔 DB 전체 스냅샷(=덤프 파일)을 다시 불러옴
- 이 순간 DB 상태는 **백업을 찍었을 때와 동일**

2.2. 이후 변경 사항 반영

- 백업 찍은 뒤에도 DB에는 계속 Insert/Update/Delete가 들어옴
- 그래서 binlog를 순서대로 다시 실행하면

  → 덤프 이후 현재까지 일어난 모든 변경이 그대로 재적용됨

⇒ **최종 상태 회복**

- 덤프 + binlog 재생을 끝내면, DB는 다시 “최신 시점”의 상태

감사합니다